PAT-NO: JP409204347A

DOCUMENT-IDENTIFIER: JP 09204347 A

TITLE: GATEWAY DEVICE

PUBN-DATE: August 5, 1997

INVENTOR-INFORMATION:
NAME
TODA, HIROYOSHI
DOI, KATSUYOSHI

ASSIGNEE-INFORMATION:

NAME	COUNTRY
SHARP CORP	N/A

APPL-NO: JP08307749

APPL-DATE: November 19, 1996

INT-CL (IPC): G06F012/00, G06F012/00 , G06F013/00

ABSTRACT:

PROBLEM TO BE SOLVED: To provide a proxy **server** device for which a **cache hit ratio is improved while a cache** file is kept fresh.

SOLUTION: In addition to a first proxy process 14 for using the **cache** file 16 and writing an access log 15, a **prefetching** process 36 for extracting the access log 15 within a period longer than the valid period of the **cache** file 16, preparing an access list 30 and periodically issuing an access request to a file object inside the list 30 and a second proxy process 34 for acquiring the file object from an external **server** in response to the access request and storing it in the **cache** file 16 are provided. The plural second proxy processes 34 can be simultaneously executed and it is convenient when priority is made lower than the first proxy process at this time.

(19)日本国特許庁（ＪＰ）　(12)公開特許公報(A)　(11)特許出願公開番号

特開平9－204347

(43)公開日　平成9年(1997)8月5日

(51)Int.Cl.⁶		識別記号	庁内整理番号	FI		技術表示箇所
G06F	12/00	545		G06F	12/00	545M
		514				514M
	13/00	357			13/00	357Z

審査請求　未請求　請求項の数16　ＯＬ　（全 50 頁）

(21)出願番号　特願平8－307749

(22)出願日　平成8年(1996)11月19日

(31)優先権主張番号　特願平7－301540

(32)優先日　平7(1995)11月20日

(33)優先権主張国　日本（ＪＰ）

(71)出願人　000005049
シャープ株式会社
大阪府大阪市阿倍野区長池町22番22号

(72)発明者　戸田　浩義
大阪府大阪市阿倍野区長池町22番22号　シャープ株式会社内

(72)発明者　土居　克良
大阪府大阪市阿倍野区長池町22番22号　シャープ株式会社内

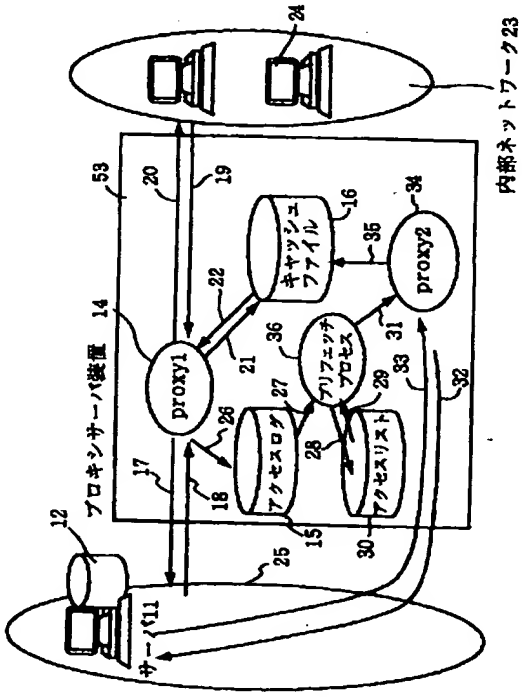
(74)代理人　弁理士　深見　久郎

(54)【発明の名称】　ゲートウェイ装置

(57)【要約】

【課題】　キャッシュファイルを新鮮に保ちながら、キャッシュヒット率の向上したproxy サーバ装置を提供する。

【解決手段】　キャッシュファイル１６を使用し、アクセスログ１５を書出す第１のproxy プロセス１４に加えて、アクセスログ１５から、キャッシュファイル１６の有効期限よりも長い期間内のアクセスログを抽出し、アクセスリスト３０を作成し、定期的にこのリスト３０内のファイルオブジェクトへのアクセス要求を発行するプリフェッチプロセス３６と、このアクセス要求に応答して外部サーバからファイルオブジェクトを取得しキャッシュファイル１６に格納するための第２のproxy プロセス３４とを含む。第２のproxy プロセス３４は複数個同時に実行されてもよく、この際に優先度を第１のproxy プロセスより低くすると好都合である。



1

【特許請求の範囲】

【請求項１】　クライアント計算機が存在する第１のネットワークと、サーバ計算機が存在する第２のネットワークとの間に介在するように設けられるゲートウェイ装置であって、

第１のネットワークのクライアント計算機からのファイルオブジェクトに対するアクセス要求に応答して、当該ファイルオブジェクトにより指定されるサーバ計算機に対して、当該ファイルオブジェクトに対するアクセス要求を行なう第１のネットワークファイル中継手段を含み、

前記第１のネットワークファイル中継手段は、

取得したファイルオブジェクトを、所定の有効期限が経過するまで一時的に蓄積するとともに、最終変更時刻をファイルオブジェクト単位で記録するためのキャッシュファイル手段と、

過去の、一定期間内に行なわれたファイルオブジェクトの転送記録を蓄積するファイル転送記録手段と、

ファイルオブジェクトに対するアクセス要求に応答して、当該ファイルオブジェクトの最終変更時刻を参照し、有効期限内のファイルオブジェクトが前記キャッシュファイル手段に存在するか否かを判断し、当該判断結果に基づいて、当該ファイルオブジェクトを前記キャッシュファイル手段、または当該ファイルオブジェクトにより指定されるサーバ計算機から取得してクライアント計算機に転送するための転送手段とを含み、

前記ゲートウェイ装置はさらに、予め定められたスケジュールに従って、前記転送記録中のファイルオブジェクトをサーバ計算機からプリフェッチして、前記キャッシュファイル手段に格納するための第２のネットワークファイル中継手段を含む、ゲートウェイ装置。

【請求項２】　前記第２のネットワークファイル中継手段は、前記第２のネットワークを介して同時に複数個のファイルオブジェクトをサーバ計算機から同時並列的にプリフェッチする、請求項１記載のゲートウェイ装置。

【請求項３】　前記第２のネットワークファイル中継手段は、同一のサーバ計算機からのプリフェッチの同時発生を回避するように、ファイルオブジェクトのプリフェッチの順序を決定する、請求項２記載のゲートウェイ装置。

【請求項４】　前記第２のネットワークファイル中継手段は、前記転送記録からファイルオブジェクトのリストを当該転送記録中の出現頻度に従って抽出し、当該リスト中のファイルオブジェクトの順序に従ってファイルオブジェクトのプリフェッチを同時並列的に行なう、請求項２記載のゲートウェイ装置。

【請求項５】　前記第２のネットワークファイル中継手段は、前記リスト中のファイルオブジェクトの順序を入替え、当該入替られた順序に従ってファイルオブジェクトのプリフェッチを同時並列的に行なう、請求項４記載

2

のゲートウェイ装置。

【請求項６】　前記第２のネットワークファイル中継手段は、前記リストを複数個のブロックに区分し、各ブロックごとにファイルオブジェクトの順序を入替え、当該順序が入替えられたリスト上での順序に従ってファイルオブジェクトのプリフェッチを同時並列的に行なう、請求項５記載のゲートウェイ装置。

【請求項７】　前記第１のネットワークファイル中継手段は、各々が第１のネットワークのクライアント計算機からのファイルオブジェクトのアクセス要求を処理するための第１の転送プロセスを複数個、同時並列的に起動させることが可能であり、

前記第２のネットワークファイル中継手段は、各々が１つの前記ファイルオブジェクトのプリフェッチを行なうための第２の転送プロセスを複数個、同時並列的に起動させることが可能であり、かつ前記第１の転送プロセスの数を検知して、前記第１の転送プロセスの数と、前記第２の転送プロセスの数との和が予め定められる上限以下となるように前記第２の転送プロセスの新たな起動を制御する、請求項２から６のいずれかに記載のゲートウェイ装置。

【請求項８】　前記第２のネットワークファイル中継手段は、予め定められた時間帯でのみファイルオブジェクトのプリフェッチを行なう、請求項１から７のいずれかに記載のゲートウェイ装置。

【請求項９】　前記第２のネットワークファイル中継手段は、前記第１のネットワークファイル中継手段の前記キャッシュファイルの有効期限以下の周期で定期的にファイルオブジェクトのプリフェッチを行なう、請求項１から８のいずれかに記載のゲートウェイ装置。

【請求項１０】　前記第１のネットワークファイル中継手段の前記キャッシュファイル手段は、前記有効期限以上の所定の消去時間が経過するまでは、当該キャッシュファイル手段に格納されているファイルオブジェクトを消去せず、

前記転送手段は、前記有効期限が経過しているが消去はされていないファイルオブジェクトに対するアクセス要求に応答して、対応するサーバ計算機にアクセスし、当該サーバ計算機内の当該ファイルオブジェクトが、前回の取得後更新されているか否かを判断し、更新されていないときには前記キャッシュファイル手段内のファイルオブジェクトを新たに取得したファイルオブジェクトとしてクライアント計算機に転送し、更新されているときには当該ファイルオブジェクトにより特定されるサーバ計算機に対して当該ファイルオブジェクトの転送要求を発して更新後の当該ファイルオブジェクトを取得し、取得したファイルオブジェクトをクライアント計算機に転送する、請求項１記載のゲートウェイ装置。

【請求項１１】　前記第１のネットワークファイル中継手段は、各々が第１のネットワークのクライアント計算

機からのファイルオブジェクトのアクセス要求を処理するための第1の転送プロセスを複数個、同時並列的に起動させることが可能であり、

前記第2のネットワークファイル中継手段は、各々が1つの前記ファイルオブジェクトのプリフェッチを行なうための第2の転送プロセスを複数個、同時並列的に起動させることが可能であり、前記第2の転送プロセスの起動時において既に稼動中の転送プロセスの数を検知して、前記稼動中の転送プロセスの数と、起動すべき第2の転送プロセスの数との和が、前記稼動中の転送プロセスの数と予め定める定数との和以下となるように前記第2の転送プロセスの新たな起動を制御する、請求項2から6のいずれかに記載のゲートウェイ装置。

【請求項12】　前記第2のネットワークファイル中継手段は、過去の一定期間中に行なったファイルオブジェクトの転送記録を、転送時に当該ファイルオブジェクトの変更が検出されたか否かを示す情報とともに蓄積するための第2のファイル転送記録手段をさらに含み、

前記第2のネットワークファイル中継手段は、前記第2のファイル転送記録手段により記録された前記転送記録中から、直前のフェッチ時において変更が検出されたファイルオブジェクトを抽出し、これらのファイルオブジェクトを、前記ファイル転送記録手段の転送記録中のファイルオブジェクトよりも優先的にサーバ計算機からプリフェッチして前記キャッシュファイル手段に格納する、請求項1に記載のゲートウェイ装置。

【請求項13】　前記第2のネットワークファイル中継手段は、予め指定された文字列パターンを記憶し、前記第2のファイル転送記録手段により記録された前記転送記録中から、前記予め定められた文字列パターンと一致する文字列を有するファイルオブジェクトを排除して抽出する、請求項10に記載のゲートウェイ装置。

【請求項14】　前記第2のファイル転送記録手段の蓄積する転送記録は、取得されたファイルオブジェクトのファイルサイズをさらに含み、

前記第2のネットワークファイル中継手段は、転送対象となるファイルオブジェクトサイズの最大値を特定する情報を予め記憶し、前記第2のファイル転送記録手段により記録された前記転送記録中から、前記最大値より大きなファイルサイズを有するファイルオブジェクトを排除して抽出する、請求項10に記載のゲートウェイ装置。

【請求項15】　前記第2のファイル転送記録手段の蓄積する転送記録は、取得が試行されたファイルオブジェクトに関するサーバ計算機の状態コードをさらに含み、

前記第2のネットワークファイル中継手段は、サーバ計算機の特定の状態コードを予め記憶し、前記第2のファイル転送記録手段により記録された前記転送記録中から、前記特定の状態コードと一致する状態コードを有するファイルオブジェクトを排除して抽出する、請求項10に記載のゲートウェイ装置。

【請求項16】　前記第1のネットワークファイル中継手段に対するクライアント計算機としての機能を実現するための手段をさらに含む、請求項1から15のいずれかに記載のゲートウェイ装置。

【発明の詳細な説明】

【0001】

【発明の属する技術分野】本発明は、ネットワーク上に分散した複数のサーバ計算機と、複数のクライアント計算機とがゲートウェイ装置（計算機）を介して通信回線で相互接続されている分散ファイルシステムに関し、特に、複数のクライアント計算機が、ゲートウェイ計算機を介してサーバ計算機内部の記憶装置内のファイルオブジェクトにアクセスする場合の分散ファイルシステムのファイル中継を行なうゲートウェイ装置に関する。

【0002】

【従来の技術】以下の説明中で「サーバ計算機」とは、ネットワーク上において何らかのサービスを提供している計算機をいい、「クライアント計算機」とは、そうしたサーバ計算機のサービスを要求し、受ける計算機をいうものとする。また「ファイルオブジェクト」とは、ファイルシステムの利用するネットワークプロトコルと、ネットワークアドレス（サーバ計算機の名称）と、ファイル名称と、ファイルの実体との組をいうものとする。ここで「ファイルオブジェクトの名称」とは、ネットワークプロトコルとネットワークアドレスとファイル名称との組のことをいう。

【0003】従来、上述したような分散ファイルシステムにおいては、複数のクライアント計算機からのサーバ計算機上のファイルオブジェクトに対する読出要求は、一旦途中のゲートウェイ計算機で中継されていた。ゲートウェイ計算機がサーバ計算機から読出したファイルオブジェクトは、クライアント計算機に中継される。それと同時にそれらファイルオブジェクトは、ゲートウェイ計算機内部のキャッシュファイル（固定ディスク、半導体メモリなど）に蓄積される。そうしたゲートウェイ計算機の一例が特開平4－313126号公報（発明の名称「分散ファイルシステムのファイル入出力方式」）に開示されている。

【0004】図12を参照して、この従来技術に開示されたシステムは、ゲートウェイ計算機110と、このゲートウェイ計算機110を介して相互に接続されたサーバ計算機100および複数のクライアント計算機120を含む。サーバ計算機100は、複数のクライアント計算機120によって共有されている。

【0005】サーバ計算機100は、ファイルを格納するディスク装置136と、ネットワークを介してディスク装置136のファイルの内容を入出力するためのファイル入出力応答手段134と、ディスク装置136内のブロック情報をネットワークを介してクライアント計算

5

機１２０に応答するためのブロック情報応答手段１３２とを含む。

【０００６】クライアント計算機１２０は、サーバ計算機１００のブロック情報応答手段１３２に対してブロック情報要求を送り、ブロック情報を受けるためのブロック情報要求手段１５２と、ゲートウェイ計算機１１０を介してサーバ計算機１００のファイル入出力応答手段１３４に対してファイル入出力要求を送り、当該ファイルを受信するためのファイル入出力要求手段１５４とを含む。

【０００７】ゲートウェイ計算機１１０は、クライアント計算機１２０（複数）とサーバ計算機１００との間に介在し、サーバ計算機１００内のディスク装置１３６と、各クライアント計算機１２０との間のキャッシュメモリとして機能するものであり、キャッシュした内容を格納するためのディスクキャッシュ１４４と、ファイル入出力要求手段１５４およびファイル入出力応答手段１３４の間に介在するキャッシュ管理手段１４２とを含む。

【０００８】図１２を参照して、このシステムは次のように動作する。まず、クライアント計算機１２０がサーバ計算機１００内のディスク装置１３６中のファイルにアクセスして１つのブロックを読込む場合の動作について説明する。ブロック情報要求手段１５０は、ゲートウェイ計算機１１０を介してサーバ計算機１００に対して読込対象のブロックにかかるブロック情報の取得を要求するブロック情報要求メッセージを発行する。

【０００９】サーバ計算機１００内のブロック情報応答手段１３２は、このメッセージに応答して、ディスク装置１３６から該当するブロック情報を取出し、そのブロック情報を有するブロック情報応答メッセージをゲートウェイ計算機１１０を介してクライアント計算機１２０内のブロック情報要求手段１５２に返却する。

【００１０】クライアント計算機１２０のファイル入出力要求手段１５０は、返却されたブロック情報に基づいて、読込対象のブロックの読込を要求するファイルアクセス要求を、ゲートウェイ計算機１１０内のキャッシュ管理手段１４２に対して発行する。

【００１１】このファイルアクセス要求を受取ったキャッシュ管理手段１４２は、そのファイルアクセス要求にかかるブロック情報と、ディスクキャッシュ１４４に記憶されている読込対象のブロックにかかるブロック情報との比較を行なう。そしてディスクキャッシュ１４４に、該当するブロック情報が存在しない場合、または両方のブロック情報の内容（更新時刻）が異なる場合（読込対象のブロックのキャッシュが有効でない場合）には、キャッシュ管理手段１４２は上述のファイルアクセス要求をサーバ計算機１００のファイル入出力応答手段１３４に対して発行する。

【００１２】サーバ計算機１００のファイル入出力応答

6

手段１３４は、このファイルアクセス要求に応答して、ディスク装置１３６内のファイルをアクセスして、該当するブロックを読込み、ゲートウェイ計算機１１０にそのブロックを転送する。

【００１３】キャッシュ管理手段１４２は、このブロックをディスクキャッシュ１４４に格納し、そのブロックにかかるブロック情報のディスクキャッシュ１４４への設定または更新を行なう。キャッシュ管理手段１４２は同時に、そのブロックをクライアント計算機１２０のファイル入出力要求手段１５４に対して転送する。

【００１４】以上でディスクキャッシュ１４４内に有効なファイルが存在しない場合のこのシステムの動作について説明した。

【００１５】次回に同一ファイルに対するアクセス要求があった場合には、ゲートウェイ計算機１１０のキャッシュ管理手段１４２は、ディスクキャッシュ１４４から当該ブロックを取出し、アクセス要求を発行したクライアント計算機１２０のファイル入出力要求手段１５４に対して当該ブロックを転送する。したがってこの場合、ゲートウェイ計算機１１０からサーバ計算機１００に対する転送要求は発行されない。したがって中継アクセスの高速化が図られる。

【００１６】類似の技術が、特開昭６３－２００２４４号公報（発明の名称「ファイル・アクセス・システム」）や、特開昭６３－２０１８４号公報（発明の名称「遠隔ファイル・アクセス・システム」）にも開示されている。いずれの技術においても、あるファイルオブジェクトに対するアクセス要求が発行された場合に、当該ファイルオブジェクトの内容が更新されているかどうかを確認するために、サーバ計算機の保持するファイルオブジェクトの変更時刻をキャッシュファイルの変更時刻と比較する。そしてキャッシュファイルの内容が古い場合には、クライアント計算機は更新されたサーバ計算機の内容を読出すようになっている。しかしこれら２件の公報に記載された技術では、キャッシュファイルはクライアント計算機内に存在しており、特開平４－３１３１２６号公報に開示されたシステムのようにシステム内にゲートウェイ計算機が含まれているわけではない。

【００１７】しかしこれら３件の従来技術文献は、サーバ計算機のファイルオブジェクトと、キャッシュされたファイルオブジェクトとの内容が食い違わないような工夫をしているという共通点を持っている。

【００１８】このように本来同一であるべきファイルの食い違いを防ぐことをインテグリティ（Integrity ）を保つという。また、食い違いの起きたファイルのデータをステールデータ（stale data）と呼ぶ。

【００１９】上述した従来の技術の基本は、ネットワーク経由でサーバ計算機の、該当するファイルオブジェクトの変更時刻を調べ、対応するキャッシュファイルの内容の変更時刻と比較するアルゴリズムを使用している。

7

そのためサーバ計算機に対するネットワーク経由の問合せを必ず行なわなければならない。ネットワークの実効的な転送速度が小さい場合や、該当するファイルオブジェクトを有するサーバ計算機にかかっている負荷が高くて即座に応答できない場合には、そうした問合せ自体にかなり時間がかかってしまうことがある。そのためネットワークファイルシステムによっては、このようなファイルキャッシュのシステムを採用していない場合もある。すなわちこうしたシステムでは、サーバ計算機の保持するファイルオブジェクトが更新されているにもかかわらず、ゲートウェイ計算機の持つキャッシュ内の対応するファイルオブジェクトは必ずしも更新されていない。このようなネットワークファイルシステム方式の代表的なものとして、いわゆるインターネットにおけるHTTP（HyperText Transfer Protocol ）を使った広域分散型マルチメディア情報システムがある。

【００２０】インターネットは、その基本プロトコルとしてTCP／IPプロトコルを利用した、グローバルなネットワークである。インターネット上に構築された地域分散型マルチメディア情報提供システムは、World Wide Web（WWW ）と呼ばれる。WWWは、ネットワーク上に分散したファイルオブジェクトを扱うことができる。これらのファイルオブジェクトは、単なるテキストデータにとどまらず、画像データ、音声データ、ビデオ画像データなどさまざまな種類のものを含む。WWWは、情報提供側にとっても、情報利用者（ユーザ）にとっても魅力的であるため、ネットワーク上のWWWに関するトラフィックが爆発的に増加している。WWWシステムでは、クライアント計算機のユーザは、グラフィカルユーザインタフェースを持ったブラウザソフトウェアをクライアント計算機上で実行させるだけで、ネットワーク上に分散したサーバ計算機の保持するさまざまなファイルオブジェクトで構成された情報を次々とアクセスすることができる。WWWは、このような操作の簡単さのために近時めざましい普及を見せている。

【００２１】そしてこのWWWシステムは、TCP／IPプロトコルの上に構築されたHTTPプロトコルでファイルオブジェクトを転送している。

【００２２】HTTPプロトコルを実施する上においては、特開平４－３１３１２６号公報に記載されたようなゲートウェイ計算機によりファイルオブジェクトを中継転送することが広く行なわれている。データはゲートウェイ装置にキャッシュされる。またWWWシステムでは、ファイルオブジェクトの中継転送においてステールデータが発生することは容認されている。すなわちゲートウェイ計算機にキャッシュされている内容が、サーバ計算機の当該ファイルオブジェクトと一致していない場合を許容している。

【００２３】キャッシュファイル内のステールデータの率をステールデータ率と呼ぶこととする。何らかの方式

8

でキャッシュファイルの内容を更新しない限り、ステールデータ率は増加する。一般にキャッシュファイルのステールデータ率が増加すると、クライアント計算機を操作しているユーザは、内容が古いと自主的に判断して、ファイルオブジェクトをサーバ計算機から再度ロードすることを試みる。この場合、キャッシュファイルの内容を利用しないプロトコルが用いられる。この結果、ゲートウェイ計算機内のキャッシュファイルシステムが意味を持たなくなる傾向がある。

【００２４】一方でこのようにステールデータの発生を容認するプロトコルの利点もある。クライアント計算機がサーバ計算機のファイルオブジェクトのアクセスをゲートウェイ計算機に中継依頼してアクセスを行なった場合、ゲートウェイ計算機のキャッシュファイルにヒットした場合には、キャッシュファイルからファイルオブジェクトが読出されてクライアントに転送される。サーバ装置に対する当該ファイルオブジェクトの変更時刻の問合せは不要である。そのため短時間にファイルオブジェクトがキャッシュから取出され、クライアントに返送される。

【００２５】HTTPプロトコルのようにステールデータを容認するネットワークファイルプロトコルは、全地球的な広域ネットワークで利用する上で有利な点を有している。すなわち、ゲートウェイ装置のキャッシュに当該ファイルオブジェクトがキャッシュされていれば、サーバ装置に対するアクセスが発生しない。その結果応答時間の短縮を図ることができるという有利な点を持っている。

【００２６】上述のようにネットワーク上のファイルオブジェクトのアクセスを高速化し、かつインターネットのトラフィックを低減させるゲートウェイ計算機は特に「proxy サーバ装置」とも呼ばれている。

【００２７】「proxy 」とは「代理人」という意味である。proxy サーバ装置はその名のとおり、クライアント計算機がネットワーク上のサーバ計算機のファイルオブジェクトにアクセスする場合に、そのアクセス要求をサーバ計算機を代理して受付け、ファイルオブジェクトの転送を中継する機能を果たす。proxy サーバ装置の概念を図１３を参照して説明する。

【００２８】図１３を参照して、proxy サーバ装置１３は、たとえばある企業内に設けられた内部ネットワーク２３と、その企業外の外部ネットワーク（インターネットなど）２５との間に介在して設けられるものである。内部ネットワーク２３は複数個のクライアント計算機２４を含み、外部ネットワーク２５は、同じく複数個のサーバ計算機１１を含んでいる。各サーバ計算機１１は、ファイルオブジェクト１２を保有している。

【００２９】proxy サーバ装置１３は、キャッシュファイル１６と、クライアント計算機からリード要求１９を受け、キャッシュファイル１６内に当該ファイルオブジ

ェクトが存在する場合にはそのファイルオブジェクトをデータ２０として返送し、キャッシュファイル１６内に当該ファイルオブジェクトの有効なものが存在しない場合には外部ネットワーク２５を介してサーバ計算機１１に対するリード要求１７を発行し、対応するファイルオブジェクトをデータ１８として受けてクライアント計算機２４に対して返送するproxy プロセス１４と、proxy プロセス１４によるファイルオブジェクトの転送記録を記録するアクセスログ１５とを含む。proxy プロセス１４は、サーバ計算機１１からファイルオブジェクトを取得したときには、当該ファイルオブジェクトをキャッシュファイル１６にも新たに書込む機能を有している。

【００３０】このproxy サーバ装置１３は次のように動作する。まず、内部ネットワーク２３内のクライアント計算機２４が、外部ネットワーク２５上のサーバ計算機１１のファイルオブジェクト１２に対するリード要求１９をproxy サーバ装置１３内のproxy プロセス１４に対して発行する。proxy プロセス１４はこのリード要求１９に応答して、キャッシュファイル１６をアクセスし、キャッシュファイル中に当該ファイルオブジェクトのキャッシュデータがあるか否かを判定する。当該ファイルオブジェクトがキャッシュされていれば、proxy サーバ装置１３固有のキャッシュファイル有効期限と、キャッシュされたファイルオブジェクトの最終変更時刻とを比較する。キャッシュが有効期限内であれば、キャッシュファイルからファイルオブジェクトを読出し（２２）、そのデータ２０を内部ネットワーク２３の、リード要求１９を発行したクライアント計算機２４に対して返送する。

【００３１】キャッシュファイル内に当該ファイルオブジェクトが存在しない場合、および存在していたとしても有効期限が過ぎている場合には、キャッシュファイル１６内に有効なファイルオブジェクトが存在しないものと判定される。その場合proxy プロセス１４は、外部ネットワーク２５のサーバ計算機１１に対して、ファイルオブジェクト１２の読出要求１７を発行する。これに応答してサーバ計算機１１は、ファイルオブジェクト１２をデータ１８としてproxy プロセス１４に返送する。proxy プロセス１４は、ログファイル１５に、アクセスログとして、ファイルオブジェクト名称（すなわちネットワークプロトコル名称と、サーバ計算機のネットワークアドレス名称と、ファイル名称との組）と、ファイルオブジェクトのデータの実体（すなわちファイルオブジェクトそのもの）と、書込時刻とを書出す（２６）。

【００３２】proxy プロセス１４はさらに、内部ネットワーク２３内のクライアント計算機２４にデータ２０を転送し、かつキャッシュファイル１６に、当該ファイルオブジェクトを書込む（２１）。

【００３３】したがってキャッシュファイル１６内には、ファイルオブジェクト名称と、ファイルオブジェクトの実体と、ファイルオブジェクトをサーバ装置から取得した時刻（最終変更時刻）とが記録されている。

【００３４】proxy サーバは、ＨＴＴＰＤや、DeleGate などのソフトウェアを、ネットワーク接続された、UNIXの動くコンピュータ上で実行することにより実現されることが一般的である。

【００３５】proxy サーバ装置の物理構成を図１４に示す。図１４を参照してproxy サーバ装置は、UNIXの動作するワークステーション２００により構成される。ワークステーション２００は、ＣＰＵ（中央処理装置）２０２と、ＣＰＵ２０２に対して内部バス２０４により接続されたメモリ２０６と、ファイル用のＩ／Ｏ（入出力）装置２０８と、ルータ２１０を介して外部ネットワークに接続される第１ネットワークＩ／Ｏインタフェース２１２と、内部ネットワークに接続される第２ネットワークＩ／Ｏインタフェース２１４とを含む。Ｉ／Ｏ装置２０８には、キャッシュファイルの蓄積、アクセスログ、および各種ワークファイルの記憶場所として使用されるファイル部２１６が接続されている。

【００３６】このワークステーション２００は、ＣＰＵ２０２が、上述したＨＴＴＰＤや DeleGate というソフトウェアを実行することによりproxy サーバ装置１３を実現する。

【００３７】図１５に、proxy サーバ装置の他の構成例を示した。このproxy サーバ装置はワークステーション３００からなっている。ワークステーション３００は、概略図１４のワークステーション２００と同様の構成であるが、図１４の第２ネットワークＩ／Ｏインタフェース２１４に代えて、モデム装置３０４に接続されるシリアルポート３０２を有している点が異なっている。そしてこのワークステーション３００は、モデム装置３０４および公衆電話回線網３０６を介して内部ネットワークと接続されている。

【００３８】図１５において、図１４と同一の部品には同一の参照番号を与えている。それらの名称および機能も同一である。したがってここではそれらについての詳しい説明は繰り返さない。

【００３９】図１６にproxy サーバ装置のさらに他の構成例を示す。図１６のproxy サーバ装置も、同じくワークステーション４００により実現される。このワークステーション４００は、概略図１４に示されるワークステーション２００と同様の構成であるが、第１ネットワークＩ／Ｏインタフェース２１２および第２ネットワークＩ／Ｏインタフェース２１４に代えて、内部ネットワーク２３に接続された１つのネットワークＩ／Ｏインタフェース４０２を有している点で異なっている。内部ネットワーク２３はルータ２１０を介して外部ネットワーク２５に接続されており、また複数個のクライアント計算機２４にも接続されている。ワークステーション４００は、内部ネットワーク２３およびルータ２１０を介して

外部ネットワーク２５のサーバ計算機１１と通信するとともに、内部ネットワーク２３およびクライアント計算機２４を介してユーザ３１と通信することができる。

【００４０】図１６と図１４とにおいて、同一の部品には同一の参照符号および名称を与えている。それらの機能も同一である。したがってここではそれらについての詳しい説明は繰り返さない。

【００４１】proxy サーバ装置は、既に述べたように中継データのキャッシュ機構を有する。すなわち、proxy サーバ装置は、その内部に存在するキャッシュファイル装置に、中継するファイルオブジェクトのデータをキャッシュする。

【００４２】proxy サーバ装置のキャッシュファイルには、proxy サーバ装置固有の有効期限が定められている。ファイルオブジェクトがキャッシュに書込まれてから有効期限内に、同じサーバ計算機上のファイルオブジェクトへのアクセス要求をクライアント計算機から受取った場合には、キャッシュファイルに以前に書込まれたファイルオブジェクトを取出してクライアント計算機に転送する。これによりサーバ計算機に対するアクセスは発生しない。

【００４３】このようなゲートウェイ装置によるキャッシュの効果を以下具体的に説明する。proxy サーバ装置をある会社組織に設置し、外部のネットワークとは６４＊

＊ｋｂｐｓの速度で接続するものとする。またこのproxy サーバは一方で、１０Ｍｂｐｓの速度の会社組織内部のローカルエリアネットワークに接続されるものとする。外部と内部のネットワークの速度差は１桁以上ある。このような例はよく見られる例であり、その典型的な例が図１６に示す構成である。

【００４４】このとき、proxy サーバで中継ファイルオブジェクトをキャッシュするものとして、キャッシュのヒット率を Hitrateとする。 Hitrateは、ファイルオブジェクトに対するアクセス要求に対してキャッシュが１００％ヒットするとき１であり、０％ヒットするとき０になる指数である。

【００４５】なおproxy サーバ装置におけるキャッシュファイルからのデータ読出およびキャッシュファイルへのデータ書込速度は、ネットワーク速度に比べて十分高速であるものとする。すなわちキャッシュファイルに対するファイルオブジェクトの入出力に要する時間は無視できるものとする。

【００４６】このとき、外部ネットワークにあるファイルオブジェクトを内部からアクセスするときの平均アクセス速度Ｖａｖｅは次の数１によって表わされる。

【００４７】

【数１】

$$Vave\ (kbps) = 64kbps \times (1 - Hitrate) + 10000kbps \times Hitrate$$

【００４８】キャッシュファイルが存在しない場合、 Hitrateは０である。したがって平均速度（ｋｂｐｓ）は、数１から外部ネットワークのアクセス速度６４ｋｂｐｓと等しくなる。また Hitrateが１であれば、すなわち１００％キャッシュファイルにヒットするのであれば、平均速度（ｋｂｐｓ）は、内部ネットワークの速度と等しく１０，０００ｋｂｐｓ（＝１０Ｍｂｐｓ）となる。つまりヒット率が高いほどクライアント計算機から見た平均アクセス速度は高くなる。ヒット率と平均アクセス速度との関係を示す数１をグラフ化したものを図１７に示す。

【００４９】ところで、キャッシュヒット率は、キャッシュファイルの量に依存している。すなわちファイルオブジェクトをなるべくたくさんキャッシュファイルの中に蓄積しておけば、再びアクセスされたファイルオブジェクトがキャッシュファイル中に含まれる確率が高くヒット率が高まる。そうした関係を示すものとして、キャッシュヒット率と、キャッシュファイルの蓄積量との関係を図１８にグラフとして示す。図１８に示すように、キャッシュファイルサイズが小さければキャッシュヒット率は０に近く、キャッシュファイルサイズを大きくすればキャッシュヒット率は徐々に上昇する。

※【００５０】proxy プロセスにおいては、キャッシュファイルの内容を維持する方式として、最初のファイルアクセスによる書込が起きてから一定期間経過したファイルオブジェクトを無効としていく制御方式がとられている。この一定期間を有効期限と呼んでいる。この有効期限は、proxy サーバ装置を管理する者が実情に合わせて適切な値を設定する。有効期限を長くとれば、キャッシュファイルに蓄積されるデータ量は増加する。その結果キャッシュのヒット率は高くなる。すなわちキャッシュのヒット率を上げるにはキャッシュの有効期限を長くとればよい。

【００５１】なお、キャッシュファイルを格納するファイル装置の記憶容量は有限であるから、有効期限を無制限に長くするわけにはいかない。有効期限が過ぎたキャッシュファイルは、何らかのタイミングで消去される。このようにキャッシュファイルを消去するまでの時間を、この明細書では消去時間と呼ぶこととする。

【００５２】本願発明者の実験によれば、キャッシュ有効期限とキャッシュファイルに蓄積されるファイル量とはほぼ比例する。この関係を図１９に示す。これは、内部ネットワークから外部ネットワークへのアクセス量が、日によらずほぼ一定であるためである。したがって

次のような関係が成り立つ。

【0053】

*【数2】

$$\text{キャッシュ有効期限} \propto \text{キャッシュファイル蓄積量} \propto \text{キャッシュヒット率}$$

【0054】この関係から、図17に示したように、キャッシュ有効期限を増やすとキャッシュのヒット率が高まることがわかる。

【0055】キャッシュ有効期限とファイルオブジェクトの量との関係は、クライアント計算機を利用する内部ユーザが、外部ネットワーク上のサーバ計算機のファイルオブジェクトをアクセスする頻度によって異なる。キャッシュ有効期限とキャッシュファイル蓄積量とは、ほぼ比例するものの、その比例係数はネットワーク速度、内部クライアント計算機ユーザ数などの使用環境により変動する。

【0056】なお、本願発明者の関係するシステムでは、キャッシュファイルが1日程度の有効期限では、キャッシュヒット率は12%程度である。有効期限を14日にするとキャッシュヒット率は41%程度になることが経験されている。有効期限14日のときにはファイルオブジェクトが700MB程度キャッシュファイルに蓄積されていることが観察されている。

【0057】

【発明が解決しようとする課題】proxy サーバ装置におけるキャッシュファイルにおいては、キャッシュヒット率をできるだけ高めるためには有効期限をできるだけ長くとり、キャッシュファイルの容量が許す限りデータを蓄積することが有効である。しかし一方で、外部ネットワーク上のサーバ計算機のファイルオブジェクトの内容は、修正変更されることがあり得る。キャッシュファイルの有効期限が長いと、その期間中は、サーバ計算機のファイルオブジェクトが内容変更されているにもかかわらず、内部ネットワークのクライアント計算機のユーザは、キャッシュファイルに蓄積された古い内容を読出すことになってしまうことになる。

【0058】たとえば天気図や電子新聞などのファイルオブジェクトをサーバ計算機上で提供する場合を考える。こうしたファイルオブジェクトは、同一名称でありながらそのデータの性格上毎日情報更新されるものが多い。たとえばキャッシュ有効期限が7日である場合を考えると、キャッシュにある天気図や電子新聞等の情報は、6／7以上の確率で古いものとなり、結局役に立たない情報となってしまう。

【0059】このようにキャッシュファイルの内容のうち、外部ネットワークに存在する元のデータが変更され、キャッシュの内容と一致していない割合をステールデータ率と呼ぶ。すなわちキャッシュ有効期限を増やせば、キャッシュ蓄積サイズが増加し、キャッシュヒット率は増加するが、有効期限が長くなった結果、キャッシュ内容のステールデータ率も増加してしまう。

【0060】したがって従来技術においては、キャッシュファイルの中のファイルオブジェクトの内容の新鮮さを保つことと、キャッシュヒット率を高く維持することとは、互いに相反する目標である。

【0061】既に述べたように一般にキャッシュファイルのステールデータ率が増加すると、クライアント計算機のユーザは、内容が古いはずであると自主的に判断し、ファイルオブジェクトデータをサーバ計算機から再度ロードするようになる。これには、キャッシュファイルを参照せず、実際にサーバ計算機からロードすることを指定するプロトコルが利用される。この結果、キャッシュが意味を持たなくなる傾向がある。

【0062】このようなステールデータ率の増加は望ましいことではなく、そのためキャッシュの有効期限はあまり長くとることはできない。通常は24時間以下である。しかし有効期限をこの程度とした場合には、キャッシュファイルにファイルオブジェクトが十分に蓄積されず、キャッシュのヒット率は高まらない。その結果平均ファイルオブジェクト転送速度を十分上げることができないという問題点があった。また、キャッシュのヒット率を高めるにあたって、他の通常プロセスに対して与える悪影響はできるだけ低くした方が好ましい。さらに、通常の他のプロセスが多数あった場合にも、キャッシュのヒット率を向上させるための作業が常にできるのが好ましく、またそのための作業自体できるだけ効率良く行なうことが望ましい。

【0063】それゆえに請求項1に記載の発明の目的は、キャッシュファイルの内容を新鮮に保つとともに、キャッシュヒット率も向上させることである。

【0064】請求項2から6記載の発明の目的は、キャッシュファイルの内容を新鮮に保つとともに、キャッシュヒット率も向上させるようにし、さらにプリフェッチに要する時間を短縮することである。

【0065】請求項7または8記載の発明の目的は、通常のユーザの利用に悪影響を与えることなく、キャッシュファイルの内容を新鮮に保つとともに、キャッシュヒット率も向上させるようにすることである。

【0066】請求項9記載の発明の目的は、常にキャッシュファイルの内容を新鮮に保つとともに、キャッシュヒット率も向上させるようにすることである。

【0067】請求項10記載の発明の目的は、無駄なトラフィックを発生させることなく、キャッシュファイルの内容を新鮮に保つとともに、キャッシュヒット率も向上させるようにすることである。

15

【0068】請求項11記載の発明の目的は、通常のユーザの利用に与える悪影響を最小限に抑えつつ、キャッシュファイルの内容を新鮮に保ちかつキャッシュヒット率をも向上させるための作業を着実に行なえるゲートウェイ装置を提供することである。

【0069】請求項12から16に記載の発明の目的は、キャッシュファイルの内容を新鮮に保つことと、キャッシュヒット率を向上させることとを、ともに、かつ効率的に達成することができるゲートウェイ装置を提供することである。

【0070】

【課題を解決するための手段】請求項1に記載の発明は、クライアント計算機が存在する第1のネットワークと、サーバ計算機が存在する第2のネットワークとの間に介在するように設けられるゲートウェイ装置であって、第1のネットワークのクライアント計算機からのファイルオブジェクトに対するアクセス要求に応答して、当該ファイルオブジェクトにより指定されるサーバ計算機に対して、当該ファイルオブジェクトに対するアクセス要求を行なう第1のネットワークファイル中継手段を含む。第1のネットワークファイル中継手段は、取得したファイルオブジェクトを、所定の有効期限が経過するまで一時的に蓄積するとともに、最終変更時刻をファイルオブジェクト単位で記録するためのキャッシュファイル手段と、過去の、一定期間内に行なわれたファイルオブジェクトの転送記録を蓄積するファイル転送記録手段と、ファイルオブジェクトに対するアクセス要求に応答して、最終変更時刻を参照し、有効期限内のファイルオブジェクトがキャッシュファイル手段に存在するか否かを判断し、当該判断結果に基づいてキャッシュファイル手段または当該ファイルオブジェクトにより指定されるサーバ計算機を選択的にアクセスして当該ファイルオブジェクトを取得してクライアント計算機に転送するための転送手段とを含む。ゲートウェイ装置はさらに、予め定められたスケジュールに従って、転送記録中のファイルオブジェクトをサーバ計算機からプリフェッチして、キャッシュファイル手段に格納するための第2のネットワークファイル中継手段を含む。

【0071】一定期間内に転送が行なわれたファイルオブジェクトは、第2のネットワークファイル中継手段によって、予め定められるスケジュールに従って、アクセス要求がなくてもプリフェッチされる。有効期限内のものはもちろん、有効期限が切れたものもプリフェッチされキャッシュファイル手段に新たに格納され有効となる。実際にアクセス要求があったときに、当該ファイルオブジェクトであって有効期限内のものがキャッシュファイルに存在している可能性が高くなる。また有効期限を長くしても、プリフェッチ時に最新のものに更新されるので、キャッシュファイル手段内のファイルオブジェクトを、高い確率でサーバ計算機のファイルオブジェクトと一致させることができる。そのため、キャッシュファイルの内容を新鮮に保つとともに、キャッシュヒット率も向上させることが可能である。

16

【0072】請求項2記載のゲートウェイ装置は、請求項1記載のゲートウェイ装置であって、第2のネットワークファイル中継手段は、第2のネットワークを介して同時に複数個のファイルオブジェクトをサーバ計算機から同時並列的にプリフェッチする。

【0073】同時並列的にプリフェッチが行なわれるので、順次行なう場合と比較してプリフェッチに要する時間を短縮することができる。

【0074】請求項3記載のゲートウェイ装置は、請求項2記載のゲートウェイ装置であって、第2のネットワークファイル中継手段は、同一のサーバ計算機からのプリフェッチの同時発生を回避するように、ファイルオブジェクトのプリフェッチの順序を決定する。

【0075】転送記録では、同一のサーバ計算機に対するアクセス要求が集中して発生することが多い。同一のサーバ計算機に対して一度にプリフェッチのためのアクセス要求が集中すると、サーバ計算機に負荷がかかる上に、通信路にも負荷がかかるため、ファイルオブジェクトの転送に時間がかかる。そこで同一のサーバ計算機からのプリフェッチの同時発生を回避するようにプリフェッチの順序を決定することで、そうした特定のサーバ計算機に負荷が集中することがなくなり、プリフェッチの速度を向上させることができる。

【0076】請求項4記載のゲートウェイ装置は、請求項2記載のゲートウェイ装置であって、第2のネットワークファイル中継手段は、転送記録からファイルオブジェクトのリストを当該転送記録中の出現頻度に従って抽出し、当該リスト中のファイルオブジェクトの順序に従ってファイルオブジェクトのプリフェッチを同時並列的に行なう。

【0077】ファイルオブジェクトのプリフェッチにあたって、転送記録中の出現頻度の高いファイルオブジェクトが優先的にプリフェッチされる。そうしたファイルオブジェクトに対するアクセス要求は将来も高い割合を示すと考えられるので、キャッシュヒット率の向上が期待できる。

【0078】請求項5記載のゲートウェイ装置は、請求項4記載のゲートウェイ装置であって、第2のネットワークファイル中継手段は、リスト中のファイルオブジェクトの順序を入替え、当該入替られた順序に従ってファイルオブジェクトのプリフェッチを同時並列的に行なう。

【0079】転送記録中には、同一のサーバ計算機に対するアクセス要求が集中して発生することが多い。同一のサーバ計算機に対して一度にプリフェッチのためのアクセスが集中すると、サーバ計算機に負荷がかかる上に、通信路上でも負荷がかかるため、ファイルオブジェ

クトの転送に時間がかかる。転送記録から抽出されたファイルオブジェクトのリスト中のファイルオブジェクトの順番を入替え、その順序でプリフェッチをすることで、同一のサーバ計算機からのプリフェッチの同時発生を回避できるようになる。したがって特定のサーバ計算機に負荷が集中することがなくなり、プリフェッチの速度を向上させることができる。

【００８０】請求項６記載のゲートウェイ装置は、請求項５記載のゲートウェイ装置であって、第２のネットワークファイル中継手段は、リストを複数個のブロックに区分し、各ブロックごとにファイルオブジェクトの順序を入替え、当該順序が入替えられたリスト上での順序に従ってファイルオブジェクトのプリフェッチを同時並列的に行なう。

【００８１】このような入替を行なうと、同一のサーバ計算機へのアクセス要求の集中的な発生が回避される上に、ファイルオブジェクトのリスト中の出現頻度順序は、ブロックのサイズの粒度で保たれる。したがって、リスト中の出現頻度の高いファイルオブジェクトのプリフェッチが、特定のサーバ計算機へのアクセス要求の集中的な発生を招くことなく行なえる。

【００８２】請求項７記載のゲートウェイ装置は、請求項２から６のいずれかに記載のゲートウェイ装置であって、第１のネットワークファイル中継手段は、各々が第１のネットワークのクライアント計算機からのファイルオブジェクトのアクセス要求を処理するための第１の転送プロセスを複数個、同時並列的に起動させることが可能である。第２のネットワークファイル中継手段は、各々が１つのファイルオブジェクトのプリフェッチを行なうための第２の転送プロセスを複数個、同時並列的に起動させることが可能であり、かつ第１の転送プロセスの数を検知して、第１の転送プロセスの数と、第２の転送プロセスの数との和が予め定められる上限以下となるように第２の転送プロセスの新たな起動を制御する。

【００８３】第１のプロセスが多数発生しているときは第２のプロセスの起動を抑制し、第１のプロセスが僅かしか発生していないときには第２のプロセスを多数起動させることができるので、通常のユーザによるファイルオブジェクトのアクセス要求が、プリフェッチのために悪影響を受けることがない。また通常のユーザによるアクセス要求の少ないときには随時プリフェッチを行なえるので、キャッシュの新鮮さを保つことができる。

【００８４】請求項８記載のゲートウェイ装置は、請求項１から７のいずれかに記載のゲートウェイ装置であって、第２のネットワークファイル中継手段は、予め定められた時間帯でのみファイルオブジェクトのプリフェッチを行なう。

【００８５】ユーザによるアクセス要求が非常に少ない時間帯にプリフェッチを行なってキャッシュファイルの新鮮さを保つことができるとともに、ユーザによるアクセス要求が発生する時間帯になるとプリフェッチを行なわないようにでき、通常のユーザによるファイルオブジェクトのアクセス要求が、プリフェッチのために悪影響を受けることがない。

【００８６】請求項９記載のゲートウェイ装置は、請求項１から８のいずれかに記載のゲートウェイ装置であって、第２のネットワークファイル中継手段は、第１のネットワークファイル中継手段のキャッシュファイルの有効期限以下の周期で定期的にファイルオブジェクトのプリフェッチを行なう。

【００８７】キャッシュファイルの有効期限以下の周期で定期的にキャッシュファイルのファイルオブジェクトのプリフェッチが行なわれるので、キャッシュファイル中のファイルオブジェクトは常に有効期限内に保たれる。その結果、プリフェッチとプリフェッチとの間の全期間にわたりプリフェッチの効果を保つことができる。

【００８８】請求項１０記載のゲートウェイ装置は、請求項１に記載のゲートウェイ装置であって、第１のネットワークファイル中継手段のキャッシュファイル手段は、有効期限以上の所定の消去時間が経過するまでは、当該キャッシュファイル手段に格納されているファイルオブジェクトを消去しない。転送手段は、有効期限が経過しているが消去はされていないファイルオブジェクトに対するアクセス要求に応答して、対応するサーバ計算機にアクセスし、当該サーバ計算機内の当該ファイルオブジェクトが、前回の取得後更新されているか否かを判断する。そして更新されていないときにはキャッシュファイル手段内のファイルオブジェクトを新たに取得したファイルオブジェクトとしてクライアント計算機に転送する。更新されているときには当該ファイルオブジェクトにより特定されるサーバ計算機に対して当該ファイルオブジェクトの転送要求を発して更新後の当該ファイルオブジェクトを取得し、取得したファイルオブジェクトをクライアント計算機に転送する。

【００８９】有効期限が経過しているファイルオブジェクトであっても、消去時間前であれば消去されない。こうしたファイルオブジェクトについてのアクセス要求が発生したときに、有効期限が切れていても、サーバ計算機において更新されていないファイルオブジェクトは、直接サーバ計算機から取得することなく、キャッシュファイルから得ることができる。したがって無駄なトラフィックの発生を抑制できる。その上、そうしたアクセス要求があって、かつサーバ計算機において当該ファイルオブジェクトが更新されていなかった場合には、実際にファイルオブジェクトをサーバ計算機から改めて取得することなくプリフェッチと同じ効果を得ることができる。

【００９０】請求項１１記載の発明に係るゲートウェイ装置は、請求項２から６のいずれかに記載のゲートウェイ装置であって、第１のネットワークファイル中継手段

は、各々が第1のネットワークのクライアント計算機からのファイルオブジェクトのアクセス要求を処理するための第1の転送プロセスを複数個、同時並列的に起動させることが可能である。第2のネットワークファイル中継手段は、各々が1つのファイルオブジェクトのプリフェッチを行なうための第2の転送プロセスを複数個、同時並列的に起動させることが可能である。第2のネットワークファイル中継手段は、第2の転送プロセスの起動時において、既に稼動中の転送プロセスの数を検知して、稼動中の転送プロセスの数と、起動すべき第2の転送プロセスの数との和が、稼動中の転送プロセスの数と予め定められる定数との和以下となるように第2の転送プロセスの新たな起動を制御する。

【0091】第1のプロセスが多数発生しているときは第2のプロセスの起動を抑制し、第1のプロセスが僅かしが発生しないときには第2のプロセスを多数起動させることができるので、通常のユーザによるファイルオブジェクトのアクセス要求が、プリフェッチのために悪影響を受けることがない。また通常のユーザによる要求の少ないときには随時プリフェッチを行なえるので、キャッシュの新鮮さを保つことができる。また、稼動中の転送プロセスの数が上限に達していても、予め定められる定数個までの第2の転送プロセスを起動することができるので、プリフェッチを着実に行なうことができる。そのため、キャッシュの新鮮さを保つことができる。またキャッシュファイル手段の容量を小さくする必要はないので、キャッシュヒット率も向上する。

【0092】請求項12に記載の発明に係るゲートウェイ装置は、請求項1記載のゲートウェイ装置であって、第2のネットワークファイル中継手段は、過去の一定期間内に行なったファイルオブジェクトの転送記録を、当該ファイルオブジェクトの変更が検出されたか否かを示す情報とともに蓄積するための第2のファイル転送記録手段をさらに含む。第2のネットワークファイル中継手段は、第2のファイル転送記録手段によりファイルオブジェクトのプリフェッチ時に記録された転送記録中から、直前のプリフェッチ時において変更が検出されたファイルオブジェクトを抽出し、これらのファイルオブジェクトを、ファイル転送記録手段の転送記録中のファイルオブジェクトよりも優先的にサーバ計算機からプリフェッチしてキャッシュファイル手段に格納する。

【0093】前回のフェッチ時に変更が検出されたファイルオブジェクトが、それ以外のファイルオブジェクトよりも優先してプリフェッチされる。こうしたファイルオブジェクトは、以後も変更される頻度が高いと考えられる。そのため、請求項1に記載の発明の効果に加えて、キャッシュファイルの内容を新鮮に保つ上で、前回のフェッチ時以降に変更があった可能性の高いファイルオブジェクトが優先的にプリフェッチされるので、プリフェッチを効率的に行なうことができる。

【0094】請求項13に記載の発明に係るゲートウェイ装置は、請求項12記載のゲートウェイ装置であって、第2のネットワークファイル中継手段は、予め指定された文字列パターンを記憶し、第2のファイル転送記録手段により記録された転送記録中から、この予め定められた文字列パターンと一致する文字列を有するファイルオブジェクトを排除して抽出する。

【0095】ファイルオブジェクトの中には、プリフェッチすることが無意味であったり、プリフェッチしておくためにかえってシステム全体の効率が低下してしまうようなファイルオブジェクトも存在する。そうしたファイルオブジェクトの中には、特定の文字列を含んでいるものとして予め知ることができるものがある。そこでこのような特定の文字列を予め指定し文字列パターンとして第2のネットワークファイル中継手段に記憶させておき、この文字列パターンと一致する文字列を有するファイルオブジェクトを排除して抽出し、残りのファイルオブジェクトのみプリフェッチする。

【0096】こうすることで、プリフェッチにより、より有用なデータをより多くキャッシュファイル手段にプリフェッチすることができるので、請求項12に記載の発明の効果に加えて、キャッシュファイルの内容を新鮮に保つことと、キャッシュヒット率を向上させることとを、効率的に達成することができる。

【0097】請求項14に記載の発明に係るゲートウェイ装置は、請求項12の記載の発明の構成に加えて、第2のファイル転送記録手段の蓄積する転送記録は、取得されたファイルオブジェクトのファイルサイズをさらに含む。第2のネットワークファイル中継手段は、転送対象となるファイルオブジェクトサイズの最大値を特定する情報を予め記憶し、第2のファイル転送記録手段により記録された転送記録中から、記憶された最大値より大きなファイルサイズを有するファイルオブジェクトを排除して抽出する。

【0098】大きなデータサイズのファイルオブジェクトは、プリフェッチに時間がかかる。通常プリフェッチの処理は、ユーザの少ない限られた時間帯に行なわれることが多い。この限られた時間帯内に、できるだけ多くの数のファイルオブジェクトのプリフェッチを完了させようとすれば、大きなファイルオブジェクトのプリフェッチは避けることが望ましい。場合によっては有限のプリフェッチ時間内に、多大なサイズのファイルオブジェクトのプリフェッチが完了せず、結果としてプリフェッチが全く行なわれないことさえあり得る。そのため、データサイズの大きなファイルオブジェクトのプリフェッチを回避することにより、有限のプリフェッチ時間内に多数のファイルオブジェクトをファイルキャッシュ手段内にプリフェッチすることができ、キャッシュファイル手段に格納された有効なデータ量を増やすことができる。結果として、キャッシュファイルの内容を新鮮に保

つことと、キャッシュヒット率を向上させることとが、効率的に行なわれる。

【0099】請求項15に記載の発明に係るゲートウェイ装置は、請求項12に記載のゲートウェイ装置であって、第2のファイル転送記録手段の蓄積する転送記録は、取得が試行されたファイルオブジェクトに関するサーバ計算機の状態コードをさらに含んでいる。第2のネットワークファイル中継手段は、サーバ計算機の特定の状態コードを予め記憶し、第2のファイル転送記録手段により記録された転送記録中から、この特定の状態コードと一致する状態コードを有するファイルオブジェクトを排除して抽出する。

【0100】サーバ計算機が、たとえばファイル送信の行なえない異常な状態にある場合、このサーバ計算機に対してファイルオブジェクトの送信を要求するのは無駄である。そのような無駄なファイルアクセスを行なわずに、他の有効なサーバ計算機に対してファイルオブジェクトのプリフェッチを行なえば、有限の時間内により多くの有効なデータをファイルキャッシュ手段内に蓄積することができる。その結果、キャッシュファイルの内容を新鮮に保ち、かつキャッシュヒット率を向上させることが効率的に行なえる。

【0101】請求項16に記載の発明に係るゲートウェイ装置は、請求項1～15のいずれかに記載のゲートウェイ装置であって、第1のネットワークファイル中継手段に対するクライアント計算機としての機能を実現するための手段をさらに含んでいる。

【0102】ゲートウェイ装置に、クライアント計算機としての機能を持たせることにより、プリフェッチしたいファイルに対するアクセス要求をクライアント計算機としての機能を通じてゲートウェイ装置に出すことができる。そのためプリフェッチされるファイルオブジェクトの量をさらに増やすことができ、キャッシュファイルの内容を新鮮に保つことと、キャッシュヒット率を向上させることとを、ともに、かつ効率的に達成することができる。

【0103】

【発明の実施の形態】

［実施の形態1］本発明が適用されるゲートウェイ装置（proxy サーバ装置と呼ぶ）は、ネットワークインタフェースを備えたUNIX OS の稼動する計算機によって、以下の説明および図面を参照すれば容易に実現可能である。そうしたproxy サーバ装置の構成の一例は図16に示したとおりである。図16のproxy サーバ装置については既に説明したので、その構成についての詳細な説明はここでは繰返さない。

【0104】通常は、ユーザ31がクライアント計算機24を使用してproxy サーバ装置（図16におけるproxy サーバ装置400）に対して、サーバ計算機11のファイルオブジェクト12を取得するようネットワーク23を経由して依頼を出す。ファイルオブジェクト12は、前述したようにプロトコル名称と、サーバ計算機のネットワークアドレス名称と、ファイルオブジェクトの名称と、ファイルデータの実体との組からなっている。

【0105】図1に、本願発明に係るproxy サーバ装置の第1の実施の形態に係る装置53のシステム構成を示す。図1を参照して、proxy サーバ装置53は、内部ネットワーク23および外部ネットワーク25に接続された第1のproxy プロセス14と、第1のproxy プロセス14が使用するキャッシュファイル16と、第1のproxy プロセス14が、アクセスログ（転送記録）としてファイルオブジェクトの名称を記録するアクセスログ15と、第1のproxy プロセス14の過去のアクセスログ10を読出してファイルのアクセスリスト30を作成するとともに、このアクセスリスト30に掲載されたファイルオブジェクトを外部ネットワーク25のサーバ計算機11からプリフェッチするための要求を発生するプリフェッチプロセス36と、プリフェッチプロセス36のプリフェッチ要求に応答して、外部ネットワーク25のサーバ11から該当するファイルオブジェクトを取得し、キャッシュファイル16に再格納することによりキャッシュファイルの内容を最新のものに保つための第2のproxy プロセス34とを含む。

【0106】第1のproxy プロセス14と、第2のproxy プロセス34と、プリフェッチプロセス36とはいずれも、たとえば図16に示すメモリ206内にプログラムとして格納され、CPU200によってこれを実行することにより実現される。

【0107】第1のproxy プロセス14が、特許請求の範囲に記載の転送手段に相当し、キャッシュファイル16がキャッシュファイル手段に相当し、アクセスログ15がファイル転送記録手段に相当する。またプリフェッチプロセス36と、アクセスリスト30と、第2のproxy プロセス34とにより第2のネットワークファイル中継手段が構成される。

【0108】第2のproxy プロセス30は、本願発明に係るゲートウェイ装置に特有のものであるが、次のような条件に従って起動されている。

【0109】（1）　第2のproxy プロセス30は、第1のproxy プロセス14と共通のキャッシュファイル16を使用する。

【0110】（2）　第2のproxy プロセス34のキャッシュ有効期限は0に設定される。すなわち第2のproxy プロセスを経由するサーバ計算機へのアクセスは、キャッシュミスヒット状態となり必ず外部ネットワーク25内のサーバ計算機に対して行なわれ、ファイルオブジェクトを取得してキャッシュファイル16に格納する。

【0111】図1に示すproxy サーバ装置53の動作について以下説明する。内部ネットワーク23内のクライアント計算機24が、このproxy サーバ装置53に対し

てネットワークアクセス要求１９を与えるものとする。第1のproxy プロセス１４が、このネットワークアクセス要求１９を受取り、まずＩ／Ｏインタフェース２０２（図１５、図１６、図１７参照）を経由してファイル装置２１６内に存在するキャッシュファイル１６をアクセスする（２１）。また第1のproxy プロセス１４は、このネットワークアクセス要求とキャッシュファイル１６内のファイルオブジェクト名称とを比較し（２２）、一致しているか否かを判定する。一致したものがあれば第1のproxy プロセス１４はさらに、当該ファイルオブジェクトのキャッシュファイル１６内の最終変更時刻を抽出し（２２）、現在の時刻と比較する。そして現在の時刻が、当該ファイルオブジェクトの最終変更時刻を起点とする有効期限内であれば、キャッシュファイル１６から当該ファイルオブジェクトを抽出し（２２）、ネットワークＩ／Ｏインタフェース４０２（図１６参照）を経由してクライアント計算機２４にファイルオブジェクトを返す（２０）。

【０１１２】該当するファイルオブジェクトのコピーがキャッシュファイル１６内に存在していない場合、または存在していてもそのキャッシュ有効期限が切れている場合には第1のproxy プロセス１４は、ネットワークＩ／Ｏインタフェース４０２（図１６参照）を経由して外部ネットワーク２５のサーバ計算機１１にファイルオブジェクト転送要求１７を送る。このときのサーバ１１は、当該ファイルオブジェクト内のネットワークアドレス名称により特定される。

【０１１３】該当するサーバ１１は、要求されたファイルオブジェクトを第1のproxy プロセス１４に返送する（１８）。すなわちこのファイルオブジェクトが第1のproxy プロセス１４により取得される。

【０１１４】第1のproxy プロセス１４は、このファイルオブジェクトをキャッシュファイル１６にその最終変更時刻とともに書込み（２１）、アクセスログ１５にこのファイルオブジェクトの名称を記録する（２６）。

【０１１５】一方、本願発明のproxy サーバ装置５３は、プリフェッチプロセス３６と第2のproxy プロセス３０とをＣＰＵにより処理走行させてファイルオブジェクトのプリフェッチを行なう点において従来のproxy サーバ装置と相違する。

【０１１６】プリフェッチプロセス３６および第2のproxy プロセス３４は次のように動作する。プリフェッチプロセス３６は、第1のproxy プロセス１４の過去のアクセスログ１５を読出し（２７）、ファイル装置内にファイルアクセスリスト３０を作成する（２８）。プリフェッチプロセス３６はさらに、このアクセスリスト３０に掲載されたファイルオブジェクトを読出し（２９）、当該ファイルオブジェクトに対するアクセス要求を第2のproxy プロセス３４に対して与える（３１）。第2のproxy プロセス３４は、このアクセス要求を外部ネットワーク２５のサーバ１１に対して与え（３２）、当該ファイルオブジェクトを取得する（３３）。取得されたファイルオブジェクトはキャッシュファイル１６内にその最終変更時刻とともに書込まれる。すなわちこれによりキャッシュファイル１６内の当該ファイルオブジェクトは、クライアント計算機２４からのアクセス要求がなくとも最新のものに書直される。また前述のとおり第2のproxy プロセス３４のキャッシュ有効期限は０に設定されているため、プリフェッチプロセス３６からアクセス要求３１が第2のproxy プロセス３４に渡されると、必ず当該ファイルオブジェクトに対するアクセス要求が第2のproxy プロセス３４によって外部ネットワーク２０に与えられる。

【０１１７】このように第2のproxy プロセス３４を起動すると、大きく次の２つの効果を得ることができる。

【０１１８】（１）　第1のproxy プロセス１４では、キャッシュファイル１６の有効期限がある値に設定されている。この有効期限をM時間であるものとする。仮にプリフェッチプロセス３６が第2のproxy プロセス３４ではなく第1のproxy プロセス１４に対してファイルオブジェクトのプリフェッチのためのアクセス要求を与えるものとすると、キャッシュファイル１６にヒットしてしまう確率が高い。これでは、外部ネットワーク２５内の元のファイルオブジェクトに対するアクセスが行なわれず、プリフェッチが有効には行なわれない。

【０１１９】一方前述のように第2のproxy プロセス３４は、そのキャッシュ有効期限が０に設定されている。そのため第2のproxy プロセスに対してアクセス要求が与えられると、必ずキャッシュミスが発生し、外部ネットワーク２５に対するファイルオブジェクトのアクセスが行なわれる（３２）。そのようにしてサーバから当該ファイルオブジェクトが取得され（３３）、取得されたファイルオブジェクトがキャッシュファイル１６に書込まれる（３５）ことによりキャッシュファイル１６内のファイルオブジェクトが最新の状態に維持される。

【０１２０】（２）　第1のproxy プロセス１４はアクセスログ１５を残す。プリフェッチプロセス３６はこのアクセスログ１５から、プリフェッチすべきファイルオブジェクト名称を抽出したアクセスリスト３０を作成する。

【０１２１】仮にプリフェッチプロセス３６が第1のproxy プロセス１４に対して、サーバ計算機内のファイルオブジェクトのプリフェッチのためのアクセス要求を与えたとすると、その結果のアクセスログがアクセスログ１５に記録されてしまう。プリフェッチプロセス３６は、次回のプリフェッチ動作においてこのアクセスログ１５をもとにアクセスすべきファイルオブジェクトのリスト３０を作成するので、その結果作成されるアクセスリスト３０には、同一のファイルオブジェクトが多重に出現してしまうことになる。そうしたアクセスリスト３

25

0によってプリフェッチを行なった場合、クライアント計算機２４からのファイルオブジェクトのアクセス要求の傾向を反映したものとは言えなくなるおそれがある。第2のproxy プロセス３４を用い、プリフェッチを行なうことにより、アクセスログ１５にはプリフェッチにらるアクセスログは書込まれないことになる。そのためアクセスリスト３０は、クライアント計算機２４からのファイルオブジェクトのアクセス要求の傾向を正確に反映したものとなる。

【０１２２】アクセスリスト３０を作成する際には、プリフェッチプロセス３６はアクセスログ１５内から、過去の一定期間（これをｎ日とする）のファイルオブジェクト名称を抽出する。ただしｎ×２４（時間）は第１のproxy プロセス１４のキャッシュ有効期限M時間よりも大きい。

【０１２３】既に図１４を参照して説明したように、通常はキャッシュ有効期限が短ければキャッシュファイルに蓄積される容量は少なく、大きくなれば大きくなる。ヒット率はキャッシュファイルに蓄積されたファイルオブジェクトの量が多いほど高くなることは、既に図１３を参照して説明した。しかしまた、キャッシュファイルサイズを大きくすればステールデータ率が増加してしまうことについても既に説明した。

【０１２４】本願発明のゲートウェイ装置では、第１の＊

26

＊proxy プロセス１４のキャッシュ有効期限はM時間である。しかしプリフェッチプロセス３６がプリフェッチの対象とするファイルオブジェクトは、ｎ日（ｎ×２４＞M）のアクセスログから抽出されている。したがってプリフェッチプロセス３６および第２のproxy プロセス３４によってキャッシュファイル１６内に蓄積されるファイルオブジェクトの量は、ｎ×２４時間分だけある。すなわち、蓄積されているファイルオブジェクトのデータ量は十分大きいのでキャッシュのヒット率は高くなる。一方で第１のproxy プロセス１４のキャッシュ有効期限はM時間に保たれているので、プリフェッチを適切な時期に行なっておけばステールデータ率が高くなるおそれはない。

【０１２５】本発明を非常に有効に適用できる例として、インターネット上のWWWシステムがある。WWWシステム上でのproxy サーバ装置において本願発明のプリフェッチプロセスを行なう手順を以下に説明する。

【０１２６】WWWシステムでは、ネットワーク上に分散したファイルオブジェクト名称はUniform Resource Locator（ＵＲＬ）と呼ばれる形式で表現され、特定される。ＵＲＬの一例を次に示す。

【０１２７】

【数３】

```
http://www.xxx.co.jp/test/index.html
```

【０１２８】数３において「http」は使用するプロトコルを示す。「www.xxx.co.jp 」は、ネットワーク上のＨＴＴＰサーバ計算機のアドレスを示すものであり、ネットワーク上で同一のものはないよう選ばれている。また「/test/index.html」はサーバ計算機内のファイル名称を示す。

【０１２９】第１のproxy プロセスとして一般に利用さ※れているDeleGateを使用する場合は、そのアクセスログの例は次のように、内部ネットワーククライアント計算機名称と、時刻と、“ＨＴＴＰプロトコル（ファイル取得要求）”、などとなる。

【０１３０】

【数４】

27 28

```
ｸﾗｲｱﾝﾄ計算機名称 [時刻] "GET http://naragw.sharp.co.jp
                /oldindex.html HTTP/1.0" 200 1717 0+0:N
ｸﾗｲｱﾝﾄ計算機名称 [時刻] "GET http://naragw.sharp.co.jp
                /sharpcolor.gif HTTP/1.0" 200 1370 0+0:N
ｸﾗｲｱﾝﾄ計算機名称 [時刻] "GET http://naragw.sharp.co.jp
                /isg.mono.gif HTTP/1.0" 200 3331 0+0:N
ｸﾗｲｱﾝﾄ計算機名称 [時刻] "GET http://naragw.sharp.co.jp
                /shoin.gif HTTP/1.0" 200 17936 0+0:N
ｸﾗｲｱﾝﾄ計算機名称 [時刻] "GET http://naragw.sharp.co.jp
                /Zaurus.gif HTTP/1.0" 200 16859 0+0:N
ｸﾗｲｱﾝﾄ計算機名称 [時刻] "GET http://naragw.sharp.co.jp
                /Prostation.gif HTTP/1.0" 200 16345 0+0:N
ｸﾗｲｱﾝﾄ計算機名称 [時刻] "GET http://naragw.sharp.co.jp
                /isg.gif HTTP/1.0" 200 20782 0+0:N
ｸﾗｲｱﾝﾄ計算機名称 [時刻] "GET http://naragw.sharp.co.jp
                /S2.gif HTTP/1.0" 200 11635 0+0:N
```

【０１３１】数４に示すアクセスログは、図２に示すような、出願人がインターネット上で公開している情報ページをアクセスした場合のログに相当する。図２に示されるページは、テキストと、６個のグラフィックデータとから構成されているので、このページに対するアクセスを１回行なうと、６個のグラフィックデータに対するアクセスも含む７つのアクセスログが形成される。すなわち上述のログにおいて第１行目はこの情報ページのテキストに対するアクセス要求であり、残りの６行は表紙に貼り込まれた出願人の会社のロゴのグラフィックデータ（sharpcolor.gif）、出願人会社内の情報システム事＊業本部と呼ばれる事業部のロゴ（isg.gif ）と、出願人会社の４つの製品の写真のグラフィックデータ（Zaurus.gif, Shoin.gif, Prostation.gif, S2.gif ）の取得要求を示している。

【０１３２】ＨＴＴＰプロトコルは、コマンド文字列（ＧＥＴなど）と、ＵＲＬと、プロトコルバージョン（「HTTP/1.0」など）から構成されている。以下はその一例である。

【０１３３】

【数５】

```
"GET http://www.xxx.co.jp/test/index.html HTTP/1.0"
```

【０１３４】ＨＴＴＰプロトコルについては、最後に掲げる表内に示された参考文献のうち、参考文献３および参考文献４に記載されている。プリフェッチプロセスの実施手順について以下に説明するが、そのための前提条件は次の（１）および（２）となっている。

【０１３５】（１） 第１のproxy プロセスは、キャッシュ有効期限Ｍ（時間）を、キャッシュログの収集期間ｎ日より短く設定する。たとえばＭ＝２４時間、ｎ＝７日とする。

【０１３６】（２） 第２のproxy プロセスは、❶キャッシュファイルを第１のproxy プロセスのものと共有し、❷キャッシュ有効期限を０に設定する、という２つの条件に従って起動される。

【０１３７】このような条件のもとに、以下に述べる手順のプリフェッチプロセスを毎日一定時刻に起動する。毎日一定時刻にプロセスを自動起動するには、ゲートウェイ装置がUNIX OS のもとで動作するのであれば、UNIX OS に備えられているタイマ機能（cron）を使用する。これは、UNIX OS に関連してよく知られている機能であるので、その詳細についてはここでは説明しない。なお、UNIX OS 以外のＯＳでも、同様の機能が提供されていることが多い。

【０１３８】図３に、本願発明に係るプリフェッチプロセスの制御の流れを示した。このプロセスを実現するためのプログラムは、たとえば図１６のメモリ２０６内に配置され、ＣＰＵ２０２によって実行される。このプロセスとネットワークおよびファイル装置２１６との入出力には、それぞれネットワークＩ／Ｏインタフェース４０２とＩ／Ｏインタフェース２０２とが使用される。

【０１３９】図３を参照して、まずステップ１で第１のproxy プロセスのアクセスログファイルをｎ日分収集する。この収集の具体的な手順は次のとおりである。

【０１４０】第１のproxy プロセスのアクセスログファイル名称は、毎日名称が変わるように第１のproxy プロセスにより設定できる。たとえばログファイル名称の末尾に、日付が入るようにすることができる。すると、ログファイルとして次のようなものが生成されるであろう。

【0141】　＊　＊【表1】

```
/usr/local/etc/delegated/10000.http.1  は７月１日のログファイル
/usr/local/etc/delegated/10000.http.2  は７月２日のログファイル
/usr/local/etc/delegated/10000.http.3  は７月３日のログファイル
/usr/local/etc/delegated/10000.http.4  は７月４日のログファイル
/usr/local/etc/delegated/10000.http.5  は７月５日のログファイル
/usr/local/etc/delegated/10000.http.6  は７月６日のログファイル
/usr/local/etc/delegated/10000.http.7  は７月７日のログファイル
/usr/local/etc/delegated/10000.http.8  は７月８日のログファイル
/usr/local/etc/delegated/10000.http.9  は７月９日のログファイル
/usr/local/etc/delegated/10000.http.10 は７月１０日のログファイル
```

【0142】最新のｎ日分のログファイルを収集するためには、これらログファイルの名称リストを日付の降順でソートすると便利である。UNIX OS のもとでは、このようなファイルの名称リストを日付の新しいものから順に並べるには/bin/ls コマンドを－ｔオプションを指定※10※して起動すればよい。そのようなコマンド例とその結果とを次に示す。

【0143】

【表2】

/bin/ls -t /usr/local/etc/delegated/10000.http.* の結果

```
'usr/local/etc/delegated/10000.http.10 は７月１０日のログファイル
/usr/local/etc/delegated/10000.http.9  は７月９日のログファイル
/usr/local/etc/delegated/10000.http.8  は７月８日のログファイル
/usr/local/etc/delegated/10000.http.7  は７月７日のログファイル
/usr/local/etc/delegated/10000.http.6  は７月６日のログファイル
/usr/local/etc/delegated/10000.http.5  は７月５日のログファイル
/usr/local/etc/delegated/10000.http.4  は７月４日のログファイル
/usr/local/etc/delegated/10000.http.3  は７月３日のログファイル
/usr/local/etc/delegated/10000.http.2  は７月２日のログファイル
/usr/local/etc/delegated/10000.http.1  は７月１日のログファイル
```

【0144】これらのファイルリストから最近のたとえば７日分のログファイル名称を抽出することにする。この場合には、UNIXに標準的に搭載されている/usr/ucb/headコマンドを用いて先頭の７行を抽出すればよい。headコマンドは、標準入力から先頭の指定行数を標準出力に出力する。そこで、ls -t コマンドの標準出力をheadコマンドの標準入力とすれば上述の最近の７日分のログファイル名称を抽出できる。このためには、ls -t のコマンドとheadコマンドとを“｜”を使って繋ぎUNIX OSに渡せばよい。このように“｜”を使って標準入出力を★接続することは、UNIXでは普通に行なわれている。なおこのような標準入出力の接続が行なえないOSのもとでこのようなデータに対する連続的な処理を行なう場合には、他の方式を使用する必要があるが、通常のＯＳではそのためのツールが装備されているであろう。

【0145】上述の接続後のコマンドとその結果とを次に示す。

【0146】

【表3】

```
/bin/ls -t /usr/local/etc/delegated/10000.http.* | head -7
```

```
/usr/local/etc/delegated/10000.http.10 は７月１０日のログファイル
'usr/local/etc/delegated/10000.http.9  は７月９日のログファイル
/usr/local/etc/delegated/10000.http.8  は７月８日のログファイル
/usr/local/etc/delegated/10000.http.7  は７月７日のログファイル
/usr/local/etc/delegated/10000.http.6  は７月６日のログファイル
/usr/local/etc/delegated/10000.http.5  は７月５日のログファイル
/usr/local/etc/delegated/10000.http.4  は７月４日のログファイル
```

【0147】次に、図3のステップ2で、プリフェッチアクセスリストを作成する。このアクセスリストは1つのファイル“workfile.txt”にまとめられるものとする。このファイルは、その1行ごとに過去のアクセスファイルオブジェクトを記録したものとなる。したがってこのファイルを作成するために、以下のような条件でプリフェッチすべきファイルオブジェクト名称を含むアクセスログを選択する。

☆【0148】（1）　HTTPコマンド文字列がファイルオブジェクトリード要求（ＧＥＴ）であり、かつ（2）　ファイルオブジェクト名称のプロトコル部分がhttpであり、かつ（3）　ファイルオブジェクト名称がプリフェッチに適するものであること（プリフェッチに適さないものを除外すること）。

【0149】以上の条件を使ってアクセスログを抽出しリストを作る。さらにファイルオブジェクト名称のフィ

31

ールドだけをこのリストから抜出し、プリフェッチすべきファイルオブジェクト名称からなるプリフェッチアクセスリストを作成する。

【0150】上述の条件（2）は、プリフェッチする意味のあるファイルオブジェクト名称のみを抽出するためである。第1および第2のproxy プロセスとして、上に述べたように本実施の形態ではDeleGateソフトウェアを使用している。このproxy ソフトウェアがキャッシュファイルを生成するプロトコルはhttpプロトコルのみである。したがってプリフェッチして意味があるのはそのようなhttpプロトコルに従ったアクセスログのみである。それ以外のプロトコルはプリフェッチしてもキャッシュ処理の対象とはならないので無意味である。

32

＊【0151】条件（3）に示すようにプリフェッチに適さないファイルオブジェクト名称として、URL中に「？」を含むものなどがある。？を含むURL表記は、サーバ計算機に対してクライアントユーザから文字列を人手で入力する場合などに使われる。そのためこの文字を含むURL表記を有するファイルオブジェクトをプリフェッチすると、予期せぬ結果を招くおそれがある。このステップ2の処理をUNIX OS のもとで行なうには、OSとともに提供されている次のようなコマンド群を使用すればよい。

【0152】

【表4】

＊

/usr/bin/grep	によるストリングパターンの抽出
/usr/bin/grep -v	によるストリングパターンの否定抽出
/usr/bin/awk	による特定のフィールド切りだし

【0153】たとえば次のコマンドを入力することにより、このステップ2の処理を経たものがファイル“workfile.txt”にリストとして出力される。

※【0154】

【表5】

※

```
cat workfile.txt | grep -v "\?" | grep "\"GET" | awk '{ print $7 }'

                                  | grep "^http://" > workfile2.txt
```

【0155】この間の経過を以下に説明する。“workfile.txt”にあるアクセスログの形式は既に述べたように次のとおりである。

★【0156】

【表6】

★

```
クラインアント計算機名称 - - [21/Aug/1995:09:27:43 +0900]

                                  "GET http://naragw.sharp.co.jp/S2.gif HTTP/1.0"
```

【0157】したがって“workfile.txt”を標準出力し、それをｇｒｅｐ　－ｖ　”＼?”に入力することで？を含まない行を抽出して標準出力に出力する。それをさらにｇｒｅｐ　”＼”ＧＥＴ”への入力とし、それによってＧＥＴコマンドであるもののみを抽出して標準出力に出力する。

【0158】さらに空白がフィールド間の区切りであると考えると、“workfile.txt”の各行の７番目のフィールドがファイルオブジェクト名称に相当するので、ａｗｋコマンドを使用してこのファイルオブジェクト名称部分のみを抽出する。さらにこのファイルオブジェクト名称部分のプロトコル部分がｈｔｔｐである行のみを抽出するために、grep ”^http://”でフィルタリングしている。なおこのコマンドパラメータのうち^は行の先頭を示している。

☆【0159】このようにステップ2でworkfile2.txt に出力されたプリフェッチアクセスリストは、内部ネットワークの複数のクライアント計算機からのサーバ計算機へのアクセス記録である。したがってこのリストには同じファイルオブジェクト名称が何度も出現する可能性がある。そこで、図3のステップ3で、このURLリストをファイルオブジェクト名称の出現頻度順にソートし、かつ同じURLについては重複しないように1つにまとめる。

【0160】UNIXでは、このようなソート作業は、OSとともに提供されている次のコマンド群を用いて行なうことができる。

【0161】

【表7】

☆

/usr/bin/sort	によるストリングパターンのソート
/usr/bin/uniq -c	による同一内容行の集約
/usr/bin/sort -n -r	によるストリングパターンの頻度順のソート

【0162】具体的には次のコマンドとなる。

◆【表8】

【0163】

33　　34

cat workfile2.txt | sort | uniq -c | sort -n -r > workfile3.txt

【0164】すなわちworkfile2.txt でURLだけになったファイルを標準出力に出力し、これをｓｏｒｔの標準入力とし、ファイルオブジェクト名称をアルファベッ＊ト順に並べる。workfile2.txt の例を次に示す。

【0165】

【表9】

```
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/sharpcolor.gif
http://naragw.sharp.co.jp/isg.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/Shoin.gif
http://naragw.sharp.co.jp/Prostation.gif
http://naragw.sharp.co.jp/S2.gif
http://naragw.sharp.co.jp/Shoin.gif
```

【0166】workfile2.txt | sort の結果は次のようにアルファベット順（ＡＳＣＩＩコード順）になる。

※【0167】

【表10】

```
http://naragw.sharp.co.jp/
http://naragw.sharp.co.jp/Prostation.gif
http://naragw.sharp.co.jp/S2.gif
http://naragw.sharp.co.jp/Shoin.gif
http://naragw.sharp.co.jp/Shoin.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/isg.gif
http://naragw.sharp.co.jp/sharpcolor.gif
```

【0168】さらにこの出力をuniq -c の標準入力に入力すると同一ファイルオブジェクト名称の行が集約され、かつ出現頻度が第１フィールドとして付加されて各レコードが出力されるので、その出力は次のようにな★る。

【0169】

【表11】

★

```
1 http://naragw.sharp.co.jp/
1 http://naragw.sharp.co.jp/Prostation.gif
1 http://naragw.sharp.co.jp/S2.gif
2 http://naragw.sharp.co.jp/Shoin.gif
3 http://naragw.sharp.co.jp/Zaurus.gif
1 http://naragw.sharp.co.jp/isg.gif
1 http://naragw.sharp.co.jp/sharpcolor.gif
```

【0170】これらの第1フィールドを数値と考え、大きいものから順にソートする。そのためにはsort -n -r の標準入力に表１１の結果を与える。この結果各レコードは出現頻度順に並べられ、その結果は次のようにな☆る。

【0171】

【表12】

☆

```
3 http://naragw.sharp.co.jp/Zaurus.gif
2 http://naragw.sharp.co.jp/Shoin.gif
1 http://naragw.sharp.co.jp/sharpcolor.gif
1 http://naragw.sharp.co.jp/isg.gif
1 http://naragw.sharp.co.jp/S2.gif
1 http://naragw.sharp.co.jp/Prostation.gif
1 http://naragw.sharp.co.jp/
```

【0172】すなわち表8のコマンドを実行した結果のworkfile3.txt というファイルは、出現頻度順に１行に１個のファイルオブジェクト名称の並んだプリフェッチアクセスリストとなる。

【0173】さらに図3を参照して、ステップ4で、プ◆50リフェッチアクセスリスト２５の１行が読出せるか否かを調べる。読出される行がない場合にはプリフェッチアクセスリストの最後であるからこの処理は終了する。読出す行がある場合にはステップ5に進む。

【0174】ステップ5では、読出した1行のファイル

35

オブジェクト名称をもとに第2のproxy プロセス34を用いて、サーバ計算機に対してネットワークアクセスをする子プロセスを起動する。この子プロセスをプリフェッチ子プロセスと呼ぶ。この子プロセスの終了を待って処理をステップ4に戻し、ステップ4とステップ5との処理を繰返すプリフェッチプロセスループを形成する。

【0175】なお、図3には示されていないが、ステップ1からステップ5の処理により生成されたプリフェッチプロセスループを、指定時刻に強制終了するようにO＊

```
echo "kill -9 プロセスID番号" | at 8:00
```

【0177】このコマンドにより、8時00分にプロセスを終了させるkill命令を発行するように、OSに対して予約をすることができる。

【0178】なおネットワークアクセス部の実現方法としては、プリフェッチプロセスから直接HTTPプロトコルを発生すればよい。しかしUNIXで一般に利用されているlynx（参考文献6参照）などのWWWクライアントプログラムを利用すると、HTTPプロトコルを発生するプログラムを新たに製作しなくともHTTPプロトコルに従ったプリフェッチ子プロセス操作を実現でき※20

```
lynx -source http://www.xxx.co.jp/test/index.html

                                    > temp_file
```

【0182】さらにproxy プロセス経由のアクセスも、UNIX OS の環境変数http__ proxyを設定することにより、lynxから利用可能である。また、proxy プロセスがDeleGateのようなproxy ソフトウェアであれば、URL表記にproxy を指定することによりproxy 経由のアクセスが可能である。これについては参考文献12を参照されたい。

【0183】後者の方式を採用する場合について以下説明する。proxy プロセスが、ネットワークアドレスがpr★

```
lynx -source http://proxyserver:10001/-_-

http://www.xxx.co.jp/test/index.html > temp_file
```

【0185】したがって本願発明を実施する際には、同様にしてproxy プロセスを第2proxy プロセスのポートに設定してやればよい。

【0186】本願発明ではこのようなファイルオブジェクト（URL）のプリフェッチにより、第1のproxy プロセスの管理するキャッシュファイルをn×24／M倍に増やし、かつ最新状態に保つことを目的としている。temp__file自体は利用されないので、このファイルは捨てても構わない。

【0187】図3のステップ1では、n日分アクセスログファイルを集めた。このnという数字は、キャッシュファイルの最大容量とファイル転送実績から定めればよ☆50

36

＊Sのタイマをセットする。OSタイマは、UNIX OS に標準的に提供されているatコマンドを利用して指定時間動作を指定することにより行なえる。より具体的には、上記したステップ1からステップ5によるプリフェッチプロセスのプロセスID番号を使用して次のコマンドをOSに与える。

【0176】

【表13】

※る。その方法を以下に説明する。

【0179】lynxはUNIX OS の上で動作するプログラムであるので、以下の説明ではUNIXの記法を使用する。

【0180】lynxでは次のコマンドを指定すれば、指定URLを読出してtemp__fileというファイルに書込むことができる。

【0181】

【表14】

★oxyserver であるゲートウェイ計算機のTCP／IPの10001番のポートを利用しているプロセスであるならば、このproxy プロセス経由でサーバ計算機のファイルオブジェクトhttp://www.xxx.co.jp/test/index.html をアクセスするためには次の表15のコマンドを指定すればよい。

【0184】

【表15】

☆い。たとえば1GBのキャッシュファイル装置に1日当り100MBのキャッシュデータが蓄積されるようであれば、n＝5程度に定めればよい。これにより、プリフェッチは500MB程度の容量となり、キャッシュファイル装置が溢れることはない。このように、キャッシュファイルの蓄積の実績を考慮することにより、キャッシュファイルを溢れさせないような数字nを容易に算出することができる。

【0188】上述のようなプリフェッチプロセスは、内部ネットワークから外部ネットワークにクライアント計算機ユーザのアクセスがあまりない時間帯を利用して起動することができる。こうすることにより、プリフェッ

チプロセスによるファイルオブジェクトアクセスが、内部ネットワークユーザによる外部ネットワークへのアクセスを妨げることがないように運用することができる。

【0189】上述の第1の実施の形態においては、プリフェッチプロセスは、クライアント計算機の利用が少なくなる21時00分より起動し翌日の8時00分には遅くとも終了するようにした。この場合、翌日に8時00分にプリフェッチプロセスがまだ動作しているときには、これを強制終了させるようにした。このようにある時刻からプリフェッチプロセスを開始し、別の時刻には強制的に終了させることにより、プリフェッチリストが大きくて定められた時刻までにプリフェッチが終了しない場合でも、クライアント計算機の利用が増加し始める時間帯にはプリフェッチプロセスは強制終了される。これにより、クライアント計算機によるproxy サーバ装置の利用が妨げられることはない。

【0190】第1の実施の形態では、第1のproxy プロセスのキャッシュ有効期限Mは24時間に設定されている。したがって夜間にプリフェッチプロセスにより更新されたキャッシュデータの内容は、翌朝から翌晩の21時までは少なくとも有効である。

【0191】上述のproxy サーバ装置を使用すると、次のような効果を得ることができる。キャッシュファイルの内容は、ユーザにとっては最近の24時間以内に更新された情報である。その一方でそうした情報が過去n日の間にアクセス要求があったファイルオブジェクトに対して保持されている。すなわちキャッシュファイルにはn日分のキャッシュデータが蓄積されている。キャッシュヒット率は、キャッシュされているデータ量が多くなれば高くなることは既に図13を参照して説明した。キャッシュファイルの蓄積期間を24時間とし、過去n日分のアクセスログを使用してプリフェッチを行なった場合、キャッシュヒット率が3倍以上となることが経験的にわかっている。

【0192】また、第1のproxy プロセスのキャッシュ有効期限M時間よりも小さいかまたは等しい時間間隔で上述のプリフェッチを行なうようにすれば、キャッシュファイルが常に有効期限内に保たれているのは明らかである。したがってプリフェッチから次のプリフェッチまでの間、プリフェッチの効果が有効に保たれる。

【0193】［実施の形態2］次に本願発明のゲートウェイ装置の第2の実施の形態を説明する。この実施の形態2は、実施の形態1と同様であるが、実施の形態1に対応するフローチャート（図3参照）のステップ3およびステップ4を改良したものである。

【0194】実施の形態1では、ステップ4において、プリフェッチアクセスリストからファイルオブジェクト名称を1つずつ取出し、第2のproxy プロセスを通じて外部ネットワークにアクセスしてファイルオブジェクトを取得している。しかしプリフェッチアクセスリストの行数が大きくなると、この方法では開始からすべてのアクセスの終了までに長時間を要する場合があり得る。

【0195】ところで、WWWシステムが利用するTCP／IPプロトコルを使用したネットワークでは、通信はパケット単位で行なわれる。そのためある計算機が、外部ネットワーク上の複数個のサーバ計算機と同時通信を行なうことが見掛け上可能である。そこで、図3に示すステップ5で起動されるプリフェッチ子プロセスは、複数個同時に起動し並列に動作させることができる。

【0196】ファイルオブジェクトアクセスは、外部ネットワーク上のさまざまな経路を通じて行なわれる。経路の途中に転送速度の小さな経路がありそこがボトルネックとなる場合があり得ることや、外部ネットワーク上のサーバ計算機自身に大きな作業負荷がかかるためにアクセス要求に即座に応答することができない場合などがあり得る。したがって実際のファイルオブジェクトの転送速度は、proxy サーバ装置と外部ネットワークとを結ぶ最初の通信路の最大転送速度よりも小さな転送速度となるのが通常である。

【0197】たとえばproxy サーバ装置と外部ネットワークを結ぶ通信路（図16における通信路27）の最大転送速度が64kbpsであっても、海外にあるサーバ計算機からの実際の転送速度がその10分の1以下であることはしばしばである。したがってproxy サーバ装置と外部ネットワークとを結ぶ最初の通信路の最大転送容量の範囲内で、同時に複数個のネットワーク接続を実施することが可能である。すなわち複数個のプリフェッチ子プロセスを起動して、複数個のファイルオブジェクトを同時に取得する。これにより通信路の転送容量が100%近く使用される状態にすることができる。

【0198】そこで、同時に複数のプリフェッチネットワーク接続を実行するプリフェッチ子プロセスの最大並列実行数を予め定めておく。この数をMAXPROCESS数とし、以下に述べる各ステップではメモリ（図16のメモリ206）内に置かれる定数である。

【0199】以下その具体的な手順について、図4を参照して説明する。図4に示されるフローチャートにおいて、ステップ1からステップ4までは、図3に示すステップ1～ステップ4とそれぞれ同じである。したがってこれらステップについては詳しい説明は繰返さない。図4では、図3のステップ5に代えてステップ5．1．2～5．2．4が設けられている。なお図4に示す動作を行なうに先立って、メモリ中にprocesses 変数を定義しておく。processes 変数は、現在バックグラウンドで走行しているプリフェッチ子プロセスの数を示す。

【0200】まずステップ5．2．1では、URLプリフェッチアクセスリストに表われたファイルオブジェクト名称の行を1つ取出す。そして第2のproxy プロセスを使ってこのファイルオブジェクトをアクセスするプリフェッチ子プロセスを、バックグラウンドプロセスとし

て1つ起動する。なおUNIX OS においてプロセスは通常バックグラウンドでも実行させることが可能である。あるプロセスをバックグラウンドで実行させるには、コマンドラインに&を付けてそのプロセスを起動すればよ＊い。具体的なコマンド例を次に示す。

【0201】

【表16】

```
lynx -source http://proxyserver:10001/-_-

http://www.xxx.co.jp/test/index.html > temp_file &
```

【0202】続いてステップ5．2．2で、バックグラウンドで走行しているプリフェッチ子プロセスの数を数え、processes 変数に代入する。

【0203】ステップ5．2．3では、processes 変数が、予め指定された最大プリフェッチ子プロセス数MAXPROCESS未満かどうかについての判定が行なわれる。processes 変数がMAXPROCESS未満であれば制御はステップ4からステップ5．2．1に戻り、さらにプリフェッチ子プロセスがバックグラウンドで起動される。

【0204】processes 変数が指定の最大プリフェッチ子プロセス数MAXPROCESS以上であれば、ステップ5．2．4で、一定時間動作を休止（sleep ）して（たとえば10秒）再びステップ5．2．2に制御を戻す。以下、ステップ5．2．2～5．2．4の処理を繰返し行ない、先行のプリフェッチ子プロセスが終了している場合には新たなプリフェッチ子プロセスを起動し、全体としてプリフェッチ子プロセスの数がMAXPROCESS以下となるようにする。

【0205】以上のような手順に従って同時処理を進めることで、プリフェッチの開始から終了までの時間を短縮することができる。そのため既に説明した指定終了時刻までの間に、より多くのファイルオブジェクトをプリフェッチすることが可能になる。実施の形態1と比較してキャッシュファイルにプリロードすることができるファイルオブジェクトの量を増やすことができるので、結果としてキャッシュヒット率を高めることができる。実験によればMAXPROCESS数として10から20の値をとれ※※ば、プリフェッチ時間を十分に短縮することができることが確認されている。

【0206】［実施の形態3］続いて実施の形態3に係るproxy サーバ装置について説明する。実施の形態3の装置は、実施の形態2の装置をさらに改良したものである。より具体的には、図4に示すステップ5．2．1からステップ5．2．4で行なわれる処理がより高速に行なわれるように、ステップ3をさらに改良したものである。その処理の流れを図5に示す。

【0207】実施の形態1および2では、プリフェッチ子プロセスをMAXPROCESS以下の範囲内で同時に起動し、並列に処理を進めることによりプリフェッチ時間の短縮を図っていた。

【0208】しかし、実施の形態1および2のステップ1からステップ3の手順で作成されたプリフェッチアクセスリストの性質として、外部ネットワーク上の同一のサーバアドレスが連続して表われることがしばしばである。これは、既に説明したようにWWWシステムにおいて、クライアント計算機側で表示されるサーバ計算機の1つの情報ページが、テキストと複数個のグラフィックオブジェクトなど、複数個のファイルオブジェクトから構成されていることが多いためである。たとえば図2に示したような情報ページは、テキストとグラフィック混じりのWWWデータである。このデータを構成するファイルオブジェクトは次の7つである。

【0209】

【表17】

http://naragw.sharp.co.jp/index.html	テキストデータ
http://naragw.sharp.co.jp/sharpcolor.gif	グラフィックデータ
http://naragw.sharp.co.jp/lsg.gif	グラフィックデータ
http://naragw.sharp.co.jp/Zaurus.gif	グラフィックデータ
http://naragw.sharp.co.jp/Shoin.gif	グラフィックデータ
http://naragw.sharp.co.jp/Prostation.gif	グラフィックデータ
http://naragw.sharp.co.jp/S2.gif	グラフィックデータ

【0210】このようにある情報ページが複数個のファイルオブジェクトで構成されている場合、これらはほぼ同時に、連続してアクセスされ、したがってプリフェッチリストにもこれらのファイルオブジェクト名称が連続して表われるであろう。すると、これらファイルオブジェクト名称を順に取出し同時に複数個取得するプリフェッチ子プロセスを起動した場合、特定のサーバのみに対するアクセスが同時に発生することになる。一般にサーバ計算機は、同時に複数のアクセスに対処可能なように★★は構成されているが、1度に多くのアクセスが集中すれば当然その応答は悪くなる。そのためこのようなアクセス方法をとれば、アクセスが集中するサーバ計算機が、プリフェッチプロセスのボトルネックとなり得る。

【0211】そこでこの実施の形態3では、プリフェッチアクセスリストを、その元々の順序そのものではなく、特定のサーバにアクセスが集中しないような順序に並べ換える。すなわちプリフェッチアクセスリスト中のファイル名称をインターリーブ処理させる。この場合、

ファイルオブジェクト名称をランダムに並べ換えれば、特定のサーバ計算機にアクセスが集中する可能性を最も低くすることができる。しかし、プリフェッチアクセスリストは元々出現頻度順で作成されており、かつ出現頻度順にファイルオブジェクトを取得した方がヒット率の向上を期待できる。そのためプリフェッチアクセスリスト上でのファイルオブジェクト名称の出現順序をある程度は保つことが望ましい。たとえばプリフェッチの終了時刻までにすべてのプリフェッチアクセスリストのファイルオブジェクトのプリフェッチが完了しない場合もあり得るだろう。そうした場合には、出現頻度の高いものが確実にプリフェッチを終了しているようにした方がよい。

【０２１２】実測によれば、このようなプリフェッチアクセスリスト中で、同一サーバアドレスが連続する数は、たかだか数十である。そこで、プリフェッチアクセスリストを、この数十という数よりもある程度大きな数の行、たとえば千行程度のブロックに分割する。そしてそれらブロック各々の内部で、ファイルオブジェクト名称の順番を入替える。そのように各ブロックごとに行が入替えられたプリフェッチアクセスリストの先頭から実施の形態２と同じようにファイルオブジェクトのプリフェッチを実行する。こうすることで、各ブロックの中では各行の順序が入替えられているので、同一サーバアドレスが連続して出現する可能性は少なくなる。一方でブロックの順番は保たれているので、最初のプリフェッチアクセスリストにあった出現頻度順は、ブロックサイズの粒度で保たれている。したがって出現頻度の高いものを優先的にプリフェッチしながら、特定のサーバ計算機＊

＊のみに同時にアクセス要求が集中するという問題を解決することができる。

【０２１３】以下この実施の形態３での処理手順を図５を参照して説明する。ブロックサイズをBLOCKSIZE 変数とする。この変数はメモリ内に置かれる。以下の処理では、順番を入替える（インターリーブする）際に、乱数を使用する。もちろんこれは簡単に実現できる例として挙げたものであって、他の方法を使用して入替え処理を行なってもよい。

【０２１４】図５に示される手順は、図４に示される実施の形態２の手順のステップ３に代えて、ステップ３．３．１および３．３．３を用いる。他のステップについては実施の形態２と同じであるので、それらについての詳しい説明はここでは繰返さない。

【０２１５】ステップ３．３．１では、プリフェッチアクセスリストを、ファイルオブジェクト名称の出現頻度順に並べ換え、かつ同じファイルオブジェクト名称については重複しないように１つにまとめる処理を行なう。この処理は図４のステップ３の処理とほぼ同様である。

【０２１６】次いでステップ３．３．２では、ファイルオブジェクト名称をインターリーブ処理して行を入替える処理を行なう。より具体的には次の手順に従う。

【０２１７】まずプリフェッチアクセスリストのすべての行に乱数［ｉ］を付与する（ｉは行番号を示す。）。乱数計算の方法としては種々考えられるが、この実施の形態では乱数は次の式に従って計算する。

【０２１８】

【数６】

$$乱数[i]=MOD\ (rand(\ )\ ,\ BLOCKSIZE\ )$$
$$+\ (\ i\ /\ BLOCKSIZE\ )*BLOCKSIZE$$

【０２１９】ただしｒａｎｄ（　）という関数は、２の３２乗の範囲の乱数を生ずる関数であるものとする。この式の第１項は、ｒａｎｄ（　）という関数で得られた乱数に対しBLOCKSIZE 変数の剰余をとったものである。したがってこの第１項は０からBLOCKSIZE －１までの範囲の乱数（整数）となる。

【０２２０】また上式の第２項は、次のような意味を持つ。プリフェッチアクセスリストを各々BLOCKSIZE 行を含むブロックに分割し、各ブロックに対して０から始まる整数のブロック番号を付けるものとする。第２項は、このようにして付けられたブロック番号にBLOCKSIZE をかけたものと等しい。したがってこれはBLOCKSIZE 数の整数倍であり、かつ０から始まって増加していく。第２項はBLOCKSIZE が３であれば０、０、０、３、３、３、６、６、６、…となる。

【０２２１】このような乱数［ｉ］をプリフェッチアク※セスリストの各行の先頭フィールドに付与する。そして各行を、第１フィールドの数の小さい順にソートし、さらに各行の先頭から乱数［ｉ］を取り除く。こうした処理によりBLOCKSIZE 行からなるブロックごとに、行がランダムにインターリーブされたプリフェッチアクセスリストが生成される。

【０２２２】上述のようにこのとき、最初のプリフェッチアクセスリストにあった出現頻度順は、ブロックサイズの粒度で保たれている。

【０２２３】以下に、乱数によるプリフェッチアクセスリスト内の各行のインターリーブの例を具体的に示す。まずオリジナルのプリフェッチアクセスリストが次のようなものであるとする。

【０２２４】

【表１８】

43　　44

http://naragw.sharp.co.jp/
http://naragw.sharp.co.jp/ZAURUS/index.html
http://pine.kuee.kyoto-u.ac.jp/
http://naragw.sharp.co.jp/Prostation.gif
http://naragw.sharp.co.jp/Shoin.gif
http://naragw.sharp.co.jp/sharpcolor.gif
http://naragw.sharp.co.jp/isg.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://www.sjmercury.com/
http://www.ntt.jp/japan/index-j.html
http://www.sjmercury.com/icons/goto.gif
http://www.sjmercury.com/icons/ntoday.gif
http://www.sjmercury.com/icons/hyprlink.gif
http://naragw.sharp.co.jp/ZAURUS/zaurus.gif
http://www.sjmercury.com/icons/main.gif
http://home.netscape.com/
http://www.sjmercury.com/icons/ntodaycl.gif
http://spectrum.eg.bucknell.edu/~boulter/crayon/
http://spectrum.eg.bucknell.edu/~boulter/crayon/crayon.gif
http://naragw.sharp.co.jp/ZAURUS/yousu.gif
http://naragw.sharp.co.jp/PRO/index.html
http://www.sjmercury.com/icons2/subscrb8.gif
http://www.sjmercury.com/grafix/home.gif

【0225】これをBLOCKSIZE =10として既に述べた式に従ってそれぞれ第1フィールドに乱数を付与すると次のようになる。

*【0226】
【表19】
*20

0	http://naragw.sharp.co.jp/
5	http://naragw.sharp.co.jp/ZAURUS/index.html
4	http://pine.kuee.kyoto-u.ac.jp/
1	http://naragw.sharp.co.jp/Prostation.gif
4	http://naragw.sharp.co.jp/Shoin.gif
9	http://naragw.sharp.co.jp/sharpcolor.gif
2	http://naragw.sharp.co.jp/isg.gif
5	http://naragw.sharp.co.jp/Zaurus.gif
6	http://www.sjmercury.com/
7	http://www.ntt.jp/japan/index-j.html
18	http://www.sjmercury.com/icons/goto.gif
11	http://www.sjmercury.com/icons/ntoday.gif
16	http://www.sjmercury.com/icons/hyprlink.gif
11	http://naragw.sharp.co.jp/ZAURUS/zaurus.gif
18	http://www.sjmercury.com/icons/main.gif
11	http://home.netscape.com/
16	http://www.sjmercury.com/icons/ntodaycl.gif
19	http://spectrum.eg.bucknell.edu/~boulter/crayon/
12	http://spectrum.eg.bucknell.edu/~boulter/crayon/crayon.gif
19	http://naragw.sharp.co.jp/ZAURUS/yousu.gif
22	http://naragw.sharp.co.jp/PRO/index.html
25	http://www.sjmercury.com/icons2/subscrb8.gif
22	http://www.sjmercury.com/grafix/home.gif

【0227】さらにこれを第1フィールドの数字の昇順にソートすると次のようなリストが得られる。この例ではBLOCKSIZE が10と小さいため、十分にインターリーブされてはいないが、各10行単位の内部で行の順番がランダムに入替えられていることがわかる。なおこの表※

※はインターリーブのための手順を示す一例にすぎないことに注意されたい。

【0228】
【表20】

0	http://naragw.sharp.co.jp/
1	http://naragw.sharp.co.jp/Prostation.gif
2	http://naragw.sharp.co.jp/isg.gif
4	http://naragw.sharp.co.jp/Shoin.gif
4	http://pine.kuee.kyoto-u.ac.jp/
5	http://naragw.sharp.co.jp/ZAURUS/index.html
5	http://naragw.sharp.co.jp/Zaurus.gif
6	http://www.sjmercury.com/
7	http://www.ntt.jp/japan/index-j.html
9	http://naragw.sharp.co.jp/sharpcolor.gif
11	http://home.netscape.com/
11	http://naragw.sharp.co.jp/ZAURUS/zaurus.gif
11	http://www.sjmercury.com/icons/ntoday.gif
12	http://spectrum.eg.bucknell.edu/~boulter/crayon/crayon.gif
16	http://www.sjmercury.com/icons/hyprlink.gif
16	http://www.sjmercury.com/icons/ntodaycl.gif
18	http://www.sjmercury.com/icons/goto.gif
18	http://www.sjmercury.com/icons/main.gif
19	http://naragw.sharp.co.jp/ZAURUS/yousu.gif
19	http://spectrum.eg.bucknell.edu/~boulter/crayon/
22	http://naragw.sharp.co.jp/PRO/index.html
22	http://www.sjmercury.com/grafix/home.gif
25	http://www.sjmercury.com/icons2/subscrb8.gif

【０２２９】このインターリーブ処理をした後に実施の形態２と同様にプリフェッチ子プロセスの並列実行を行なう。同一のサーバ計算機をアドレスとして含むファイルオブジェクトがプリフェッチアクセスリスト中に連続して表われることが避けられるため、特定のサーバ計算機へアクセスが集中することが避けられる。その結果各サーバ計算機に負荷が集中することなく、サーバ計算機から迅速に必要なファイルオブジェクトをプリフェッチすることができ、結果としてプリフェッチ処理全体の時間を短縮化することができる。キャッシュファイルにプリフェッチされるキャッシュデータの量を増やすことができるので、キャッシュヒット率を高めることができる。

【０２３０】［実施の形態４］続いて実施の形態４のゲートウェイ装置であるproxy サーバ装置について説明する。この実施の形態４は、実施の形態３をさらに改良したものである。実施の形態３では、プリフェッチ子プロセス数としてMAXPROCESS数を最大とするようなプリフェッチ子プロセス数の起動の制御を行なっていた。しかしこうした処理では、たとえばクライアント計算機からのファイルオブジェクトへのアクセス要求が多く発生しているときにも上記したMAXPROCESS数だけのプリフェッチ子プロセス数が起動されてしまうことがあり得、その結果クライアント計算機によるファイルオブジェクトへのアクセス要求に対する応答が低下してしまうおそれがある。この実施の形態４は、そうした問題を解決するための改良である。

【０２３１】実施の形態４では、クライアント計算機のユーザが第１のproxy プロセスを利用して外部ネットワークのサーバ計算機に多数アクセスするような時間帯には、プリフェッチ子プロセスが起動される数を少なくすることにより、内部ユーザによるネットワーク利用活動を妨げないようにしている。

＊【０２３２】具体的な処理手順を図６に示す。図６のフローチャートは、図５に示すフローチャートのステップ５．２．１～５．２．４に代えてステップ５．４．１～５．４．４を採用したものであり、動的にプリフェッチ子プロセスのプロセス数を制御する点に特徴がある。他のステップは、第１～第３の実施の形態において説明したものとそれぞれ同じであるので、ここではその詳細な説明は繰返さない。

【０２３３】図６を参照して、ステップ５．４．１で、プリフェッチアクセスリストに現れたファイルオブジェクト名称から１行を抽出する。そして第２のproxy プロセスを使ってそのファイルオブジェクトにアクセスするためのプリフェッチ子プロセスをバックグラウンドプロセスとして１つ起動する。

【０２３４】続いてステップ５．４．２で、バックグラウンドで走行しているプリフェッチ子プロセスの数と、第１のproxy プロセスにより起動されているネットワークアクセス中継子プロセスの数を調べ加算して、processes 変数に代入する。

【０２３５】続いてステップ５．４．３で、processes 変数が、予め指定された最大プリフェッチ子プロセス数MAXPROCESSより小さいか否かについての判定が行なわれる。小さければ制御はステップ４に戻り、上述した新たなプリフェッチ子プロセス起動の処理が行なわれる。

【０２３６】processes 変数が、予め指定されたMAXPROCESS以上であれば制御はステップ５．４．３からステップ５．４．４に進み、一定時間（たとえば１０秒）休止してステップ５．４．２に制御を戻す。

【０２３７】図６に示すような処理手順に従って、この実施の形態４のproxy サーバ装置は動作する。プリフェッチ子プロセスの個数と、第１のproxy プロセスを利用している内部ユーザのネットワークアクセスの中継子プロセス数との「合計」が一定となるように、プリフェッ

チ子プロセスが新たに起動されるのを制御する。第1のproxy プロセスを利用しているユーザによるネットワークアクセスの子プロセスがMAXPROCESS数以上であれば、プリフェッチ子プロセスは新たには起動されず起動待ちとなる。

【0238】内部のユーザが第1のproxy プロセスを多数利用している時間帯に上述のプリフェッチプロセスを実行しても、プリフェッチ子プロセスには低い優先権しか与えられないので、内部ユーザのネットワーク利用活動を妨げないようにすることができる。プリフェッチプロセスを、内部ユーザの活動時間帯に始動させても、内部ユーザのネットワーク利用は妨げられない。またプリフェッチプロセスが、クライアント計算機のユーザの活動時間に食い込んでもやはり内部ユーザのネットワーク利用は妨げられない。

【0239】その結果、プリフェッチプロセスの開始から終了までより長時間を使用してプリフェッチを行なうことができる。そのため多くのファイルオブジェクトをプリフェッチすることが可能になり、したがってキャッシュデータの量を増やすことができる。上述のキャッシュデータの量とキャッシュヒット率との関係から、結果としてキャッシュヒット率を高めることができる。

【0240】これは、「数1」に示された式を参照して、ヒット率を高めるので、平均ファイルオブジェクト＊転送速度を増加させるという効果を奏する。別の見方をすれば、このようにすることにより同じ平均ファイルオブジェクト転送速度を維持しながら、内部ネットワークのより多くのユーザの利用を可能にするということができる。

【0241】［実施の形態5］次に、実施の形態5のゲートウェイ装置の一例であるproxy サーバ装置について説明する。既に延べた実施の形態1～4では、夜間など内部ユーザが外部ネットワークを利用しない時間帯に、n日分（＝n×24時間＝N時間）のプリフェッチを実行した。キャッシュファイルについては、有効期限をM時間であるとすると、M時間以上古いキャッシュファイルについては定期的に消去していた。こうした古いキャッシュファイルの消去については上述の実施の形態1～4では言及していない。

【0242】UNIX OS であれば、一定時間以上変更のないファイルはｆｉｎｄコマンドを利用して消去可能である。下に挙げるのは、毎日一定時刻に起動されるキャッシュ消去コマンドの一例である。cache というディレクトリの下にある、1日以上変更のない古いファイルが、このキャッシュ消去コマンドにより消去される。

【0243】

【表21】

```
find /cache/ -mtime +1 -a -exec rm -rf {} \;
```

【0244】ところで、実施の形態1～4のproxy サーバ装置では、午後9時から翌朝午前8時までの間にプリフェッチを実施していた。これにより夜間のトラフィックが、昼間の内部ユーザによるアクセスのn倍となる。

【0245】実際には、キャッシュが有効期限を過ぎている場合には、DeleGateのような一般的に使用されているproxy プロセスにおいては、HTTPに従って次のようなアルゴリズムが取られる。

【0246】（1） 外部ネットワークのサーバ計算機に存在するファイルオブジェクトの取得要求（GET命令）を出すときに、GET命令にキャッシュファイルの時刻Tcache を付けて要求を出す。

【0247】（2） この要求を受けたサーバ計算機は、その持っている対応するファイルオブジェクトの最終変更時刻Torg とTcache とを比較する。

【0248】（3） そしてTorg ＞Tcache のとき、すなわちファイルオブジェクトの最終変更時刻よりもキャッシュファイルの時刻のほうが古いときのみ、サーバ計算機は当該ファイルオブジェクトをproxy サーバ装置に対して転送する。すなわち、前回キャッシュされた時刻以降に元のファイルオブジェクトが変更されている場合のみ当該ファイルオブジェクトの転送が実行される。

【0249】（4） Torg ＞Tcache ではない場合には、サーバ計算機は当該キャッシュファイルを使えという応答コードをproxy サーバ装置に返答する。

※【0250】上述のアルゴリズムによれば、実際には元のデータのうち変更されているものだけがサーバ計算機からproxy サーバ装置に転送される。変更されていない場合には、キャッシュファイルの当該ファイルオブジェクトの最終変更時刻が、そうした確認がされた時刻に更新される。

【0251】実施の形態5では、proxy プロセスのこのようなアルゴリズムを利用することにより、ネットワーク上のデータ転送量を減少させる。そのためこの実施の形態5のproxy サーバ装置では、キャッシュファイル（有効期限M時間）のキャッシュデータのうちで、M時間以上古いものを定期的に必ず消去してしまうのではなく、M＜N≦Eであるような時間Eを定め、これ以上古いファイルオブジェクトのみを定期的に消去するようにしている。これにより有効期限の過ぎたキャッシュファイルのファイルオブジェクトであっても、直ちには消去されず、E時間内は有効に活用することができる可能性がある。たとえばサーバ計算機に対するプリフェッチを行なった際に、元データが変更されていないことが確認されたデータは、改めてサーバ計算機からproxy サーバ装置に転送する必要はない。キャッシュされているデータと元データとは一致しているためである。このとき、キャッシュファイルへの書込時刻のみが更新される。結果としてこのデータについては、プリフェッチが行なわれた時刻にサーバ計算機から新たに転送されたのと同じ

効果を得ることができ、プリフェッチ時のネットワークのデータ転送量を減少させることができる。

【0252】ここで、N＝Eとすると管理が簡単である。時間Eとしては、キャッシュファイルが溢れない程度であってかつなるべく大きな値を採用すればよい。N＝Eとして、Nを主体に考えてこれを大きく取る場合を考える。Nを大きく取れば、既に延べたようにプリフェッチされるべきファイルオブジェクトが増えるので、キャッシュファイルの蓄積量は、増大するはずである。しかし実際にはプリフェッチは一定時刻に強制終了されるので、実施の形態1～4のようにプリフェッチにおいて必ずデータ転送を行なっても、キャッシュファイルの蓄積量はNとともに増大するわけではない。ところがこの実施の形態5の装置では、変更されていないファイルオブジェクトについてはデータ転送を行なうことなく、かつプリフェッチを行なったのと同じ効果を得ることができる。したがって、キャッシュファイルの蓄積量は、実は時間Eの大小に影響される。

【0253】実際の測定結果によれば、E／24時間＝n＝7日のときに、全プリフェッチデータのうち6％程度しかファイルオブジェクトのキャッシュへの再ロードを引起こしていない。キャッシュファイルの最終変更時刻情報のついた「条件付きGET命令」による最終変更時刻情報の転送量は、実際のファイルデータの転送と比較して量的に無視し得るほど小さい。そのためこの実施の形態5に従ったプリフェッチにより増加するトラフィックは、サーバ計算機のファイルオブジェクトのうち、キャッシュに格納された時刻以後に更新されたものについてのデータ転送に対するものだけである。実測例によればプリフェッチによるトラフィックの増加は7日分×0．06＝0．42日分である。

【0254】この実施の形態5のproxy サーバ装置で行なわれるプリフェッチ処理によるキャッシュのヒット率向上により、昼間のユーザの活動による外部ネットワークから内部ネットワークへのファイルオブジェクト転送量は、約20％減少していることが観察されている。したがって上述したプリフェッチによるトラフィックの増加を考慮しても、1日のトータルで見れば、プリフェッチによる外部ネットワークからのデータ流量は量的に著しく増えているわけではない。したがって外部ネットワークのトラフィックをいたずらに増やすことなく、キャッシュファイルの内容を新鮮に保つことができるという利点が得られる。

【0255】［実施の形態6］ところで、以上の実施の形態でのアクセスログの例を数4に示してある。数4に示したアクセスログは、第1のproxy プロセスとしてDeleGateソフトウェアを使用する場合のアクセスログ例であるが、これらのうち既に説明したもの以外の部分についてここで説明する。これらは、サーバ計算機の状態コード、ファイルオブジェクトのデータサイズ、キャッシュ状況などである。

【0256】サーバ計算機の状態コードとは、サーバ計算機がクライアント計算機からの要求に対して発する3桁の数字で表わされる応答である。状態コードの最初の数字は、状態コードのカテゴリを示しており、1、2、3、4、5が使用される。200番代の状態コードはクライアント計算機の要求を問題なく理解したことを示す。たとえば状態コードが200の場合には、要求が正常に受付けられた旨を示す。300番代の状態コードはサーバ計算機が再転送の必要があることを示し、400番代の状態コードはクライアント計算機のエラーを示し、500番代の状態コードはサーバ計算機のエラーを示している。また、100番代の状態コードは将来のために予約されており、現在は使用されていない。

【0257】ファイルオブジェクトのデータサイズとは、ファイルオブジェクトのサイズをバイト数で示している。キャッシュ状況とは、サーバ側データとキャッシュ内容との関係を示すものである。キャッシュ状況が「H」ならキャッシュヒット、「N」ならキャッシュが存在せずに新たにキャッシュファイルにファイルオブジェクトが書込まれたこと、「O」であればサーバ側データがキャッシュ内容よりも新しいということを示す。

【0258】本発明の実施の形態6のゲートウェイ装置では、このようなファイルオブジェクト転送記録を利用する。

【0259】実施の形態6のゲートウェイ装置は、実施の形態1～5をさらに改善したものである。既に述べた本発明のゲートウェイ装置では、利用される際にはクライアント計算機による利用が少ない時間帯でのプリフェッチの実行を行なうことが想定されている。しかしそれら時間は夜間など、限られた時間に限定される。そのためプリフェッチできるファイルオブジェクトの量には制限があり、プリフェッチ対象のファイルオブジェクトをすべてプリフェッチすることができないことがある、という問題点がある。実施の形態6のゲートウェイ装置はこうした問題を解決すべく、短い時間に効率良くファイルオブジェクトをプリフェッチすることを目的としている。

【0260】図7に、本願発明の実施の形態6に係るゲートウェイ装置の一例であるproxyサーバ装置54の概略のブロック図を、proxy サーバ装置54が接続されるネットワーク25および内部ネットワーク23とともに示す。図7において、図1に示すproxy サーバ装置53と同一の部品には同一の参照符号および名称を付している。それらの機能も同一である。したがって、ここではそれらについての詳しい説明は繰返さない。

【0261】図7に示されるproxy サーバ装置54が図1に示されるproxy サーバ装置53と異なるのは、第2のproxy プロセス34が、アクセスログ39にプリフェッチの伝送記録を書込むこと（40）と、アクセスリス

ト３０から読出したファイルオブジェクトに対して第２のproxy プロセス３４によるプリフェッチを行なわせる点においては図１に示されるプリフェッチプロセス３６と同様であるが、この新たに設けられたアクセスログ３９の内容を参照し、アクセスログに含まれるファイルの変更状況を表わす情報に従って、前回のプリフェッチ時に変更が検出されたファイルオブジェクトをアクセスリスト３０からのファイルオブジェクトに優先してプリフェッチするように第２のproxy プロセス３４を制御するためのプリフェッチプロセス４１を含んでいることである。

【０２６２】第１のproxy プロセス１４、第２のproxy プロセス３４およびプリフェッチプロセス４１がいずれもメモリ中に置かれ、ＣＰＵによって処理されるのは実施の形態１の装置と同様である。

【０２６３】このproxy サーバ装置５４では、第２のproxy プロセス３４は、ファイルオブジェクトのプリフェッチの結果のアクセスログ３９を記録する。プリフェッチプロセス４１は、このアクセスログ３９を走査し、前回のプリフェッチの結果、プリフェッチされたファイルオブジェクトが、その前にプリフェッチされた時点以後に更新されていたものの名称を抽出し、変更検出リストを作成する。変更検出リストを、名称順、出現頻度順に並べることにより同一名称のものを取除いた上で、アクセスリスト３０から抽出されたプリフェッチリストの前部にマージする。そしてこのマージされたプリフェッチリストに従って第２のproxy プロセス３４のプリフェッチプロセスを実行させる。

【０２６４】前回のプリフェッチ時に、その前のプリフェッチ以後変更されていたファイルオブジェクトは、以後も比較的頻繁に変更される可能性が高い、ということができる。したがって上述のように作成されたプリフェッチリストは、変更の可能性の高いファイルオブジェクトが前部に置かれたものである。そのためこれらは、規定時間のうちでも最初の方でプリフェッチされる。変更の可能性の低いものは、これらの後にプリフェッチされることになるが、変更の可能性が低いので、既にフェッチされているデータで十分である可能性が高い。そのため限られた時間内に、変更可能性の高いファイルオブジェクトを優先的にプリフェッチすることができ、キャッシュファイル１６の内容を新鮮に保つことができる。

【０２６５】以下、プリフェッチプロセス４１の動作を詳しく述べるが、その概略についてまとめておくと次のようになる。

【０２６６】（１）　第２のproxy プロセス３４のアクセスログ３９から、過去の一定の日数分のログを集める。サーバ計算機側のファイルオブジェクトが、前回のプリフェッチの結果その前のプリフェッチ以後変更されていたという情報を持っているファイルオブジェクトのリストを、変更検出リストとして作成する。これらは、前回のプリフェッチの結果、キャッシュファイルに書込まれたファイルオブジェクトのリストでもある。

【０２６７】（２）　第１のproxy プロセスのアクセスログファイル３０をもとに、過去の一定期間のファイルオブジェクト名称を抽出し、プリフェッチアクセスリストを作成する。このプリフェッチアクセスリスト自体は実施の形態１におけるものと同様である。

【０２６８】（３）　（２）で作成されたプリフェッチアクセスリストの先頭部分に、（１）で作成された変更検出リストをマージする。

【０２６９】（４）　このようにして作成されたプリフェッチアクセスリストに基づき、第２のproxy プロセス３４を利用して、外部ネットワークのサーバ計算機から、プリフェッチアクセスリストにあるファイルオブジェクトをプリフェッチする。

【０２７０】（５）　これによりキャッシュファイル１６内に、第２のproxy プロセス３４によってファイルオブジェクトが書込まれ、最新に維持される。このとき第２のproxy プロセス３４は、アクセス結果をアクセスログ３９に書込む。

【０２７１】図８および図９を参照して、このproxy サーバ装置５４の動作をより詳細に説明する。なお図８および図９に示されるフローチャートの各ステップをたとえば「step１」、「step５」などと記載することにする。

【０２７２】step１

第２のproxy プロセスのアクセスログファイルを、最近のＰ日分収集する。なおアクセスログファイルは、各日別に別々のファイルとして作成されるものとする。このログファイルはプリフェッチによるアクセスログである。したがって、プリフェッチ時にサーバ計算機のファイルオブジェクトにアクセスした結果に従って、ファイルオブジェクトが更新されたかどうかという情報を有している。なお発明者の経験では、Ｐは２程度で十分である。

【０２７３】第２のproxy プロセスのアクセスログファイル名称は、毎日名称が変わるように第２のproxy プロセスにより設定することができる。たとえばログファイル名称の末尾に日付が入るようにすることができる。その場合ファイル名称をたとえば”/usr/local/etc/delegated/10000.http.日”という名称にすることができる。例として７月１日から１０日までのログファイルの名称を示すと次のようになる。

【０２７４】

【表２２】

53　　54

```
/usr/local/etc/delegated/10001.http.1   7月1日のログファイル
/usr/local/etc/delegated/10001.http.2   7月2日のログファイル
/usr/local/etc/delegated/10001.http.3   7月3日のログファイル
/usr/local/ tc/delegated/10001.http.4   7月4日のログファイル
/usr/local/ tc/delegated/10001.http.5   7月5日のログファイル
/usr/local/etc/delegated/10001.http.6   7月6日のログファイル
/usr/local/etc/delegated/10001.http.7   7月7日のログファイル
/usr/local/etc/delegated/10001.http.8   7月8日のログファイル
/usr/local/etc/delegated/10001.http.9   7月9日のログファイル
/usr/local/etc/delegated/10001.http.10  7月10日のログファイル
```

【0275】最新のログファイルから順番に並べるため　＊【0276】
に、UNIX OS の下では、/bin/ls コマンドを「-t」オプ　10　【表23】
ションで起動する。具体的には次のとおりである。　＊

```
/bin/ls -t /usr/local/etc/delegated/10001.http.*
```

【0277】これにより次の結果を得ることができる。　※【表24】
【0278】　※

```
/usr/local/etc/delegated/10001.http.10  7月10日のログファイル
/usr/local/etc/delegated/10001.http.9   7月9日のログファイル
/usr/local/etc/delegated/10001.http.8   7月8日のログファイル
/usr/local/etc/delegated/10001.http.7   7月7日のログファイル
/usr/local/etc/delegated/10001.http.6   7月6日のログファイル
/usr/local/etc/delegated/10001.http.5   7月5日のログファイル
/usr/local/etc/delegated/10001.http.4   7月4日のログファイル
/usr/local/etc/delegated/10001.http.3   7月3日のログファイル
/usr/local/etc/delegated/10001.http.2   7月2日のログファイル
/usr/local/etc/delegated/10001.http.1   7月1日のログファイル
```

【0279】既に述べたようにP＝2で十分であるから、このようにして並べ変えられたファイルリストから最近の2日分のログファイル名称を抽出する。そのためには、UNIX OS に標準的に搭載されている/usr/ucb/head コマンドを用いる。headコマンドは標準入力から先頭の指定行数を標準出力に出力するので、"｜"を使い、★

★ls -t の標準出力をheadコマンドの標準入力とすればよい。そうした標準出力の接続を行なったコマンドを次の表25に示す。
【0280】
【表25】

```
/bin/ls -t /usr/local/etc/delegated/10001.http.* | head -2
```

【0281】上のコマンドを実行すると次の結果が得られる。　☆【0282】
☆　【表26】

```
/usr/local/etc/delegated/10001.http.10  7月10日のログファイル
/usr/local/etc/delegated/10001.http.9   7月9日のログファイル
```

【0283】step2
ここで、step1で収集した第2のproxy プロセスのアクセスログファイルを1つのファイル"obsoletelist.txt"にまとめる。なおこの名称は一例にすぎず、運用に応じて適当に定めることができる。
【0284】このように"obsoletelist.txt"というファ　40
イルにアクセスログファイルをまとめるとき、ファイルオブジェクトがproxy サーバ装置54を介してユーザに中継された時刻と、プリフェッチを行なう時刻との間隔に応じた重み付けを加える。すなわち、最新のアクセス◆

◆ログファイルに、古いアクセスログファイルよりも大きな重み付けを加える。たとえば、単純にログファイルの日付に注目して次のようにすることができる。表25に示したコマンドの実行の結果得られる出力は、日付の新しいログファイルから順番に並んでいる。そのためUNIX OS に標準的に搭載されているcat コマンドを利用して次のように"obsoletelist.txt"ファイルを作成することができる。
【0285】
【表27】

```
cat "[step1]の出力の1行目のファイル" \
    "[st p1]の出力の1行目のファイル" \
    "[step1]の出力の2行目のファイル" > obsoletelist.txt
```

【0286】表27のようにcat コマンドを利用することにより、プリフェッチを行なう当日に残されたアクセスログは2回分とカウントされるのに対して、前日分の＊50

＊アクセスログは1回分とカウントされる。したがって最近アクセスされたファイルオブジェクトの出現頻度に、より大きな重み付けが与えられたアクセスログリストが

55

残される。

【0287】step3

上述のstep2で出力されたファイル"obsoletelist.txt"の各行は、過去のプリフェッチによるアクセスファイルオブジェクトの記録である。そこで、以下のような条件に従ったファイルオブジェクト名称を含むアクセスログを選択する。

【0288】・httpコマンド文字列がファイルオブジェクトリード要求（GET ）であり、かつ

・ファイルオブジェクト名称のプロトコル部分がhttpであるもの

・特定の文字列パターンにマッチするものは除外

・ファイルオブジェクト名称がプリフェッチに適さないものを除外

・ファイルオブジェクトコードのデータサイズが巨大なものを除外

・サーバ計算機の状態コードが特定のものを除外

・ファイルオブジェクトのアクセスの結果、キャッシュが更新されたものをプリフェッチ

以上の条件を使ってファイルオブジェクトを抽出する。

このようにして抽出されたリストは、前回のそのファイ

56

＊ルオブジェクトのアクセスの結果キャッシュが更新されたもののみを含むから、これをファイルオブジェクトの変更検出リストと呼ぶことができる。この変更検出リストに対して、まず出現頻度順のソートを行なった上で、同一名称のファイルオブジェクトをまとめる。またファイルオブジェクト名称のフィールドだけを取出して、プリフェッチすべきファイルオブジェクト名称からなるプリフェッチアクセスリストを作成する。

【0289】プリフェッチに適さないと考えられるファイルオブジェクト名称として、URL中に"？"を含むものがある。"？"を含むURL 表記は、クライアントユーザが文字列を手操作により入力してサーバ計算機に対して渡す場合などに使われる。そのためこのURL 表記を含む名称のファイルオブジェクトをプリフェッチすると、ユーザに代わって機械が勝手に文字を入力したことになってしまうので、有害である。

【0290】このようなファイルオブジェクト名称の抽出作業は、UNIX OS で標準で提供されている次のようなコマンド部すなわち

【0291】

【表28】

/usr/bin/cat	によるファイルの標準出力への出力。
/usr/bin/grep	によるストリングパターンの抽出。
/usr/bin/grep -v	によるストリングパターンの否定抽出。
/usr/bin/fgrep -v -f file.txt	によるストリングパターンの否定抽出。指定ストリングパターンはファイルfile.txtに格納しておく。
/usr/bin/awk	による特定のフィールド切りだし。
/usr/bin/sort	によるテキスト行のソート。

※【0293】

【表29】

grep_datasize DATASIZE	データサイズがDATASIZE以下のストリングパターンの抽出
grep_response file.txt	サーバ計算機の状態コードが特定のストリングパターンの抽出

【0292】および、C言語などを用いて記述したプログラム

【0294】などを使って行なうことができる。なお表28および表29において、左側にはコマンドおよびパラメータと、プログラム読出のためのコマンドが、右側にはそれぞれに対応する機能が、それぞれ示されている。

【0295】たとえば次のコマンドを投入することにより、step3の処理を経たリストが標準出力に出力される。

【0296】

【表30】

```
cat obsoletelist.txt \
| grep 'OS' \
| grep -v "\?" \
| gr p "\"GET" \
| grep_datasize DATASIZE |
| grep_response respons.txt |
| awk '{ print $7 }' | grep "^http://"
```

★【0297】以下、その経過について説明する。"obsoletelist.txt"にあるログファイルの形式は既に述べたように次のようになっている。

【0298】

【表31】

57 58

クラインアント計算機名称 〔時刻〕 "GET http://naragw.sharp.co.jp/52.gif HTTP/1.0"
サーバ計算機の状態コード ファイルオブジェクトのデータサイズ キャッシュ状況

【0299】末尾のキャッシュ状況は、第2のproxy プロセスのサーバ計算機へのアクセス状況によって変化する。具体的には次のとおりである。

【0300】「0 」：サーバ計算機のファイルオブジェクトの方がキャッシュのファイルオブジェクトより新しかった場合

「S 」：サーバ計算機のファイルオブジェクトとキャッシュのファイルオブジェクトとの最終変更時刻の比較により、両者が同じものであると判明した場合キャッシュ状況はこの他にもさまざまなあるが、これらのキャッシュ状況に関する情報は、プリフェッチを中継した第2のproxy プロセスによりアクセスログ39に書き出されている。そこで、"obsoletelist.txt"を標準出力にcat コマンドで出力し、それをgrep '0$' に入力することで、行末が「0 」であるものが標準出力に出力される。なおgrepというコマンドもUNIX OS に標準的に搭載されているものである。さらにこの出力を処理するためには、既に述べたように" | " を使い、この標準出力を次のコマンドの標準入力とすればよい。

【0301】その標準出力を"grep -v" ＼?"の標準入力とすることで" ? " を含まない行を抽出し、標準出力に出力する。その出力を"grep"＼"GET" の標準入力に入力する。その結果、コマンドが「get 」である行のみが抽出され、標準出力に出力される。

【0302】このようにして得られた標準出力をgrep__datasize DATASIZE の標準入力に与える。この場合、DATASIZEには、予め指定されたファイルオブジェクトのデータサイズの上限値が設定されている。この値はたとえば実際のコマンドでは具体的な値に設定されている。このコマンドの標準出力として、ファイルオブジェクトのデータサイズが定数 DATASIZE よりも小さい行のみが抽出され標準出力に得られる。

【0303】さらにこの標準出力を、grep__response response.txt の標準入力に与える。ここで、"response.txt"は、予め抽出されるべき、サーバ計算機の状態コードが記述されているファイルである。このコマンドの実行により、サーバ計算機の状態コードが"response.txt" に記述されている状態コードと一致する行のみが抽出され標準出力に出力される。

【0304】空白を区切りと考えると、表31に示されるアクセスログでは、7番目のフィールドがファイルオブジェクト名称に相当する。そこで上述したgrep__responseresponse.txt の出力を、awk' {print $7} ' の標準入力に与えてファイルアクセスログからファイルオブジェクト名称の部分のみを抽出する。

【0305】さらに、ファイルオブジェクト名称部分のプロトコル名称がhttpである行のみを抽出する必要がある。そこで上述した出力をさらに"grep " ^http://"の標準入力に与える。なおここで" ^ " は行の先頭を示す。以上のような処理の結果が標準出力に出力される。なおこのようにして得られた標準出力は、次のstep4で再び標準入力として使用される。

【0306】step4

次に、step3で出力されたファイルオブジェクトのリストから、プリフェッチに好ましくないと思われるものを取除く。具体的には、プリフェッチするのに好ましくないと思われるファイルオブジェクトが有すると思われる文字列パターンを、除外パターンリストとして"exclude.txt" というファイルに予め格納しておく。このリストの追加変更は、ゲートウェイ管理者が行なう。

【0307】具体的には、プリフェッチしてわざわざキャッシュを更新する必要はないと判断されるような内容を含むファイルオブジェクト、サイズが巨大なため、プリフェッチすると他のファイルオブジェクトのプリフェッチの妨げになるようなファイルオブジェクト、またはサーバ計算機に異常があり、プリフェッチする上で支障があるファイルオブジェクトなどが考えられる。この"exclude.txt" ファイルの記載内容の例を表32に示す。もちろん表32に示すものは仮定の例である。表32に示される文字列パターンと一致する文字列を有するファイルオブジェクトは、更新対象リストから取除かれる。

【0308】

【表32】

http://www.undisirable.co.jp/image/
http://www.too-slow.co.jp/files/

【0309】ところで、step3で標準出力に出力された情報は、複数回にわたるプリフェッチアクセスの記録であるから、同じファイルオブジェクト名称が何度も出現する場合がある。そこで、このリストをさらに出現頻度の降順にソートする。このとき同時に、同じURL については重複しないよう1つにまとめる。UNIX OS では、このようなソート作業は標準的に提供されている次のコマンド群を使って行なうことができる。

【0310】

【表33】

/usr/bin/fgr p -v -f	による特定文字パターンの排除
/usr/bin/sort	によるストリングパターンのソート
/usr/bin/uniq	による同一内容行の集約
/usr/bin/sort -n -r	によるストリングパターンの第1フィールドの数字の大きいものからのソート

59　　60

【0311】そこで、次のコマンドにより上述のソート作業を行なう。

```
[step3]の標準出力 | fgr p -v -f exclude.txt | sort \
| uniq -c | s rt -n -r  | awk '{print $2}' > obsolete_url.txt
```

【0312】

【表34】

【0313】このコマンドの処理内容は次のとおりである。まずfgrep -v -f exclude.txtでは、"exclude.txt" ファイルに含まれる文字列パターンと一致しないファイルオブジェクトだけが標準出力に出力される。さらにこれをsortの標準入力とすることで、ファイルオブジェクト名称をアルファベット順に並べる。なおこの例では、単純に、URL が完全一致すべき文字列パターンのみを"exclude.txt" ファイルで指定しているが、以下に示すように、URL が部分一致するべき文字列パターンを除外するための"exclude.txt" ファイルを用意することも考えられる。

※【0314】

【表35】

```
http://www.undisirable.co.jp/image/
http://www.too-slow.co.jp/files/
http://*cgi-bin*
http://*.mov
```

【0315】この例では、" *" は任意の文字列パターンを示している。そのため以下に示すURL も、除外対象となる。

【0316】

【表36】

```
http://www.ibmlink.japan.co.jp:80/cgi-bin/pres.sh

http://www.disney.com/DisneyVideos/masterpiece/shelves/
                                winniethepooh/Videos/clip1.mov
```

【0317】文字列パターンが部分一致するものを除外するためには、表34に示したコマンドをそのまま利用することはできない。しかし、UNIX OS に標準で用意されているコマンド群やC言語で記述したプログラム等の組合せによってこれを容易に実現することが可能である。

【0318】ここでは、表32に示す内容が"exclude.t★xt" ファイルに格納されており、表34に記載された処理コマンドが処理に用いられたとして以下説明を進める。step3の標準出力が以下のとおりであるものとする。

【0319】

【表37】

```
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/
http://www.too-slow.co.jp/files/movies.mov
http://www.undisirable.co.jp/images/picture.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/sharpcolor.gif
http://naragw.sharp.co.jp/isg.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/Shoin.gif
http://naragw.sharp.co.jp/Prostation.gif
http://naragw.sharp.co.jp/S2.gif
http://naragw.sharp.co.jp/Shoin.gif
```

【0320】fgrep -v -f exclude.txt の処理により、"exclude.txt" ファイルに格納された除外パターンリストに含まれる文字列パターンに一致する文字列を有する第3行と第4行とが取除かれる。その結果次のよう☆な標準出力が得られる。

【0321】

【表38】

```
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/sharpcolor.gif
http://naragw.sharp.co.jp/isg.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/Shoin.gif
http://naragw.sharp.co.jp/Prostation.gif
http://naragw.sharp.co.jp/S2.gif
http://naragw.sharp.co.jp/Shoin.gif
```

【0322】さらにこれをsortの標準入力に与え、ASCI◆50◆I コード順にソートするとその結果は次のようになる。

61　　62

【0323】　＊　＊【表39】

```
http://naragw.sharp.co.jp/
http://naragw.sharp.co.jp/Prostation.gif
http://naragw.sharp.co.jp/S2.gif
http://naragw.sharp.co.jp/Shoin.gif
http://naragw.sharp.co.jp/Shoin.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/Zaurus.gif
http://naragw.sharp.co.jp/isg.gif
http://naragw.sharp.co.jp/sharpcolor.gif
```

【0324】さらにこの出力をuniq -c の標準入力に与えると、同一ファイルオブジェクト名称の行が集約され、出現頻度が第1フィールドに挿入されて出力され　※

【0325】

【表40】

```
1 http://naragw.sharp.co.jp/
1 http://naragw.sharp.co.jp/Prostation.gif
1 http://naragw.sharp.co.jp/S2.gif
2 http://naragw.sharp.co.jp/Shoin.gif
3 http://naragw.sharp.co.jp/Zaurus.gif
1 http://naragw.sharp.co.jp/isg.gif
1 http://naragw.sharp.co.jp/sharpcolor.gif
```

【0326】得られたリストの各行の第1フィールドは、出現頻度の数値と考えることができる。そこでこの出現頻度の大きいものから順にソートするために、表40の内容をsort -n -rの標準入力に与えてソートするこ★

【0327】

【表41】

```
3 http://naragw.sharp.co.jp/Zaurus.gif
2 http://naragw.sharp.co.jp/Shoin.gif
1 http://naragw.sharp.co.jp/sharpcolor.gif
1 http://naragw.sharp.co.jp/isg.gif
1 http://naragw.sharp.co.jp/S2.gif
1 http://naragw.sharp.co.jp/Prostation.gif
1 http://naragw.sharp.co.jp/
```

【0328】step5

次に第1のproxy プロセスのアクセスログファイルをn日分収集する。この収集方法の具体的な手順は次のとおりである。第1のproxy プロセスのアクセスログファイル名称は、毎日名称が変わるように第1のproxy により設定できる。たとえばログファイル名称の末尾に日付が入るようにすると、第1のproxy プロセスのアクセスロ☆

【表42】

```
/usr/local/etc/delegated/10000.http.1  は7月1日のログファイル
/usr/local/etc/delegated/10000.http.2  は7月2日のログファイル
/usr/local/etc/delegated/10000.http.3  は7月3日のログファイル
/usr/local/etc/delegated/10000.http.4  は7月4日のログファイル
/usr/local/etc/delegated/10000.http.5  は7月5日のログファイル
/usr/local/etc/delegated/10000.http.6  は7月6日のログファイル
/usr/local/etc/delegated/10000.http.7  は7月7日のログファイル
/usr/local/etc/delegated/10000.http.8  は7月8日のログファイル
/usr/local/etc/delegated/10000.http.9  は7月9日のログファイル
/usr/local/etc/delegated/10000.http.10 は7月10日のログファイル
```

【0331】このファイルの名称リストを、日付の新しいものから順に並べる。そのためには、UNIX OS の下では、/bin/ls コマンドを-tオプションで起動すればよ◆

【0332】

【数7】

```
/bin/ls -t /usr/local/etc/delegated/10000.http.*
```

【0333】これにより次の結果が得られる。　＊【表43】

【0334】

＊50

☆グファイル名称は"/usr/local/etc/delegated/10000.http. 日"という名称にして毎日変わるようにすることができる。

【0329】そうするとたとえば、7月1日から7月10日までのログファイルの名称が次のとおりになる。

【0330】

◆い。

63　　64

'usr/local/etc/delegated/10000.http.10 は7月10日のログファイル
/usr/local/etc/delegated/10000.http.9　は7月9日のログファイル
/usr/local/etc/delegated/10000.http.8　は7月8日のログファイル
/usr/local/etc/delegated/10000.http.7　は7月7日のログファイル
/usr/local/etc/delegated/10000.http.6　は7月6日のログファイル
/usr/local/etc/delegated/10000.http.5　は7月5日のログファイル
/usr/local/etc/delegated/10000.http.4　は7月4日のログファイル
/usr/local/etc/delegated/10000.http.3　は7月3日のログファイル
/usr/local/etc/delegated/10000.http.2　は7月2日のログファイル
/usr/local/etc/delegated/10000.http.1　は7月1日のログファイル

【0335】さらにこれらのファイルリストから最近の7日分のログファイル名称を抽出する。そのためには、UNIX OS に標準的に搭載されている/usr/ucb/head コマンドを用いて先頭の7行を抽出すればよい。headコマンドは、標準入力に与えられるリストから、先頭の指定行数を抽出して標準出力に出力するものである。

【0336】

【数8】

```
/bin/ls -t /usr/local/etc/delegated/10000.http.* | head -7
```

【0337】この結果は次のとおりである。　※【表44】

【0338】　※

```
/usr/local/etc/delegated/10000.http.10 7月10日のログファイル
/usr/local/etc/delegated/10000.http.9 7月9日のログファイル
/usr/local/etc/delegated/10000.http.8 7月8日のログファイル
/usr/local/etc/delegated/10000.http.7 7月7日のログファイル
/usr/local/etc/delegated/10000.http.6 7月6日のログファイル
/usr/local/etc/delegated/10000.http.5 7月5日のログファイル
/usr/local/etc/delegated/10000.http.4 7月4日のログファイル
```

【0339】step6

ここで、step5で収集した第1のproxy プロセスのアクセスログファイルを、1つのファイル"workfile.txt"にまとめる。このときに、各ファイルオブジェクトが最後にユーザ中継された時刻と、プリフェッチが行なわれる時刻との間隔に応じた重み付けを各ファイルオブジェクトに与える。たとえば、単純にログファイルの日付に注目すると、表43には、新しい日付のログファイルから順番に並んでいることがわかる。そこで、UNIX OS に標準的に搭載されているcat コマンドを利用して次の処理を行なう。

【0340】

【表45】

★

```
cat "出力の1行目のファイル" \
    "出力の1行目のファイル" \
    "出力の1行目のファイル" \
    "出力の1行目のファイル" \
    "出力の1行目のファイル" \
    "出力の1行目のファイル" \
    "出力の1行目のファイル" \
    "出力の2行目のファイル" \
    "出力の2行目のファイル" \
    "出力の2行目のファイル" \
    "出力の2行目のファイル" \
    "出力の2行目のファイル" \
    "出力の2行目のファイル" \
    "出力の3行目のファイル" \
    "出力の3行目のファイル" \
    "出力の3行目のファイル" \
    "出力の3行目のファイル" \
    "出力の3行目のファイル" \
    "出力の4行目のファイル" \
    "出力の4行目のファイル" \
    "出力の4行目のファイル" \
    "出力の4行目のファイル" \
    "出力の5行目のファイル" \
    "出力の5行目のファイル" \
    "出力の5行目のファイル" \
    "出力の6行目のファイル" \
    "出力の6行目のファイル" \
    "出力の7行目のファイル" > workfile.txt
```

【0341】こうすれば、プリフェッチを行なう当日に残されたアクセスログは7回分として、前日に残されたアクセスログは6回分として、2日前に残されたアクセスログは5回分として、というように、最近にアクセスされたファイルオブジェクトの出現頻度がより高くなるようなアクセスログリストが残される。

【0342】step7

step6で収集された"workfile.txt"ファイルは、1行ご

65

とに過去のアクセスファイルオブジェクトを記録したファイルである。そこでこの各行から、以下のような条件でプリフェッチすべきファイルオブジェクト名称を含むものを選択する。

【0343】・httpコマンド文字列がファイルオブジェクトリード要求（GET）であり、かつ
・ファイルオブジェクト名称のプロトコル部分がhttpであるもの
・特定の文字列パターンにマッチするものを除外
・ファイルオブジェクト名称がプリフェッチに適さないものを除外
・ファイルオブジェクトのデータサイズが巨大なものを除外
・サーバ計算機の状態コードが特定のものを除外

＊
```
/usr/bin/grep        ストリングパターンの抽出
/usr/bin/grep -v     ストリングパターンの否定抽出
/usr/bin/awk         特定のフィールド切出
```

【0347】と、C言語などを用いて記述したプログラム

※【0348】

※　【表47】

```
grep_datasize DATASIZE    データサイズがDATASIZE以下の
                          ストリングパターンの抽出
grep_response file.txt    サーバ計算機の状態コードが特定の
                          ストリングパターン抽出
```

【0349】を使って行なうことができる。以下に、上述の抽出処理を行なうコマンド例を示す。

【0350】

【表48】

```
cat workfile.txt \
| grep -v "\?" \
| grep "\"GET" \
| grep_datasize DATASIZE \
| grep_response respons.txt \
| awk '{print $7 }' | grep "^http://"
```

【0351】上のコマンドにより行なわれる処理の経過を説明する。"workfile.txt"ファイルにあるログの形式は既に述べたように表31に示すとおりである。そこで、標準出力として得られる"workfile.txt"を"grep -v" ＼?"に入力することで" ?" を含まないようログを抽出し、標準出力に出力する。それをさらに"grep"＼"GET" の標準入力に与えることで、コマンドがget コマンドである行のみが抽出され標準出力に出力される。

【0352】"grep"＼"GET" の標準出力をgrep__datasize DATASIZE の標準入力に与えると、ファイルオブジェクトのデータサイズが、予め指定しておいた定数DATASIZEより小さい行のみが抽出され標準出力に出力される。標準出力をgrep__response response.txt の標準入力に与えると、サーバ計算機の状態コードが、予めファイル"response.txt"に記述されている状態コードと一致す★

66

＊以上の条件を使い、"workfile.txt"ファイルからファイルオブジェクト名称を抽出したリストを作成する。さらに、作成されたリストの各行のファイルオブジェクト名称のフィールドだけを取出し、プリフェッチすべきファイルオブジェクト名称のリストを標準出力に出力する。

【0344】ところで、ファイルオブジェクト名称がプリフェッチに適さないものとしては、たとえばURL 中に" ?" を含むものなとがある。これについては既に述べたとおりである。

【0345】UNIX OS では、上に述べたようなファイルオプション名称の抽出作業は、OSとともに提供されているコマンド群

【0346】

★る行のみが抽出され、標準出力に出力される。

【0353】さらに空白を区切りと考えると、各行の7番目のフィールドがファイルオブジェクト名称に相当する。そこで、awk'{print $7}' を使用してファイルオブジェクト名称部分のみを抽出する。さらにファイルオブジェクト名称のプロトコル部分が"http"の行のみを抽出するために"grep" ^http://"でフィルタリングしている（" ^" は行の先頭を示す。）。

【0354】step8

step7で標準出力に出力されたリストは、内部ネットワークの複数のクライアント計算機からの、サーバ計算機へのアクセスの記録である。したがってこのリスト中には同じファイルオブジェクト名称が何度も出現する場合がある。

【0355】そこでこのURL リストを出現頻度順にソートし、かつ同じURL については重複しないように1つにまとめる。また、step4同様、好ましくないファイルオブジェクト名称を取除くため、ファイル"exclude.txt"に格納された各行の文字列パターンと一致する文字列を有する行を削除する。UNIX OS の下では、このようなソート作業は、OSとともに提供されている次のようなコマンド群を用いて行なうことができる。

【0356】

【表49】

67	68
/usr/bin/fgrep -v -f	特定文字パターンに一致する行の削除
/usr/bin/sort	ストリングパターンのソート
/usr/bin/uniq	同一内容行の集約
/usr/bin/sort	ストリングパターンの頻度順のソート

【0357】これらコマンド群を用いてたとえば次のようなコマンドにより処理を行なう。

＊【0358】

＊　【数9】

```
|ステップ5の標準出力 | fgrep -v -f exclude.txt | sort \
| uniq -c | sort -n -r | awk '{ print $2}' > workfile2.txt
```

【0359】このコマンドは、前のステップの処理の標準出力をsortの標準入力とすることにより、ファイルオブジェクト名称をアルファベット順に並べる処理である。

【0360】なお、ファイル"exclude.txt"に、ファイルオブジェクト名称のうちURLが部分一致する文字列パターンを格納し、この文字列パターンと部分一致するURLを有するファイルオブジェクト名称を除外することも※※可能である。しかしここでは、ファイル"exclude.txt"には、URLが完全一致すべき文字列パターンが格納されているものとする。

【0361】step7の標準出力の内容が次のとおりであるものとする。

【0362】

【表50】

```
http://www.sharp.co.jp/News.gif
http://www.sharp.co.jp/
http://www.sharp.co.jp/News.gif
http://www.sharp.co.jp/sharpcolor.gif
http://www.sharp.co.jp/allsharp.gif
http://www.sharp.co.jp/News.gif
http://www.sharp.co.jp/Shoin.gif
http://www.sharp.co.jp/Prostation
http://www.sharp.co.jp/S2.gif
http://www.sharp.co.jp/Shoin.gif
```

【0363】すると、"step7の標準出力|sort"の結果は、次のように表50の各行がアルファベット順にソートされたものとなる。

【0364】

【表51】

```
http://www.sharp.co.jp/
http://www.sharp.co.jp/Prostation
http://www.sharp.co.jp/News.gif
http://www.sharp.co.jp/News.gif
http://www.sharp.co.jp/News.gif
http://www.sharp.co.jp/S2.gif
http://www.sharp.co.jp/Shoin.gif
http://www.sharp.co.jp/Shoin.gif
http://www.sharp.co.jp/allsharp.gif
http://www.sharp.co.jp/sharpcolor.gif
```

【0365】さらにこの出力を"uniq -c"の標準入力に与えると、同一ファイルオブジェクト名称を有する行が集約され、かつ同一ファイルオブジェクト名称の行の出現頻度が第1フィールドに挿入された次のようなリストが出力される。

【0366】

【表52】

★

```
1 http://www.sharp.co.jp/
3 http://www.sharp.co.jp/Prostation
1 http://www.sharp.co.jp/News.gif
1 http://www.sharp.co.jp/S2.gif
2 http://www.sharp.co.jp/Shoin.gif
1 http://www.sharp.co.jp/allsharp.gif
1 http://www.sharp.co.jp/sharpcolor.gif
```

【0367】このリストの各行の第1フィールドは、対応するURLの出現頻度を示す数値と考えることができる。そこでこの第1フィールドを数値として扱って、降順にソートするためにsort -n -rの標準入力に与える。これにより各行を出現頻度順に並べることができる。その結果は次のようになる。

【0368】

【表53】

```
3 http://www.sharp.co.jp/News.gif
2 http://www.sharp.co.jp/Shoin.gif
1 http://www.sharp.co.jp/allsharp.gif
1 http://www.sharp.co.jp/sharpcolor.gif
1 http://www.sharp.co.jp/S2.gif
1 http://www.sharp.co.jp/Prostation
1 http://www.sharp.co.jp/
```

★

【0369】すなわち、数8として記載したコマンドの出力は"workfile2.txt"ファイルとなるが、このファイル"workfile2.txt"は、出現頻度順に1行に1個のファイルオブジェクト名称がリストされたプリフェッチアク

セスリストとなる。

【0370】step9次に、step4の結果得られたファイル"obsoletelist.txt"と、step8の結果得られたファイル"workfile2.txt" とを連結して、1つのファイル"prefetch _url.txt"とする。その連結の順序は次のとおりである。

【0371】（1） 変更検出リスト"obsoletelist.tx*t"

（2） プリフェッチアクセスリスト"workfile2.txt"

この処理は、UNIX OS の下では具体的には次のコマンドを用いて行なうことができる。

【0372】

【数10】

```
cat obsolete_url.txt workfile2.txt > prefetch_url.txt
```

【0373】step10

続いて、プリフェッチリストのインターリーブ処理を行なう。このインターリーブ処理とは、実施の形態3で既に説明した処理と同じものであり、プリフェッチアクセスリストを、その元々の順序そのものではなく、特定のサーバにアクセスが集中しないような順序に並べ変える処理をいう。こうすることにより、特定のサーバ計算機にアクセスが集中する可能性を最も低くすることができる。その詳細は既に実施の形態3で説明してあるので、ここでは繰返さない。

【0374】step11

続いて、並列ネットワーク中継個数の最大定数を計算す※る。すなわち、次のステップ以降に示すプリフェッチ子プロセス起動ループで使用する、並列実行の子プロセス数の許容される最大値である数MAXPROCESSを計算する。これまで説明した実施の形態では、並行プリフェッチの同時実行の最大値であるMAXPROCESSは、予め定められた固定のパラメータであった。

【0375】すなわち、これまでの実施の形態では、プリフェッチプロセスの並列実行の子プロセスの数は次の式によって規制されていた。

【0376】

【数11】

$$\text{“全てのPROXY ネットワーク中継プロセス数”} \leq \text{MAXPROCESS}$$

【0377】このようにして、起動されるプリフェッチプロセスの並列実行の子プロセスの数を規制しておくことで、他のプロセスに対するプリフェッチ子プロセスの優先度が下がる、という効果があることはこれまでの実施の形態の説明において既に述べたとおりである。なおすべてのproxy ネットワーク中継プロセス数の中には、プリフェッチ子プロセスが含まれている。

【0378】ところで、この場合次のような問題がある。たとえばMAXPROCESS＝20と定められていたものとする。この場合、次のステップのループ実行を始めた時点で、クライアント計算機ユーザによるネットワーク中継プロセスが20個以上存在している場合がある。するとそれだけで既に、上の数11で示される条件が満たされなくなる。すなわち、

【0379】

【数12】

$$\text{“全てのPROXY ネットワーク中継プロセス数”} \leq 20$$

【0380】が成立せず、そのため新たにプリフェッチ子プロセスを起動することができない。こうした状況が連続すれば、実質的にプリフェッチを行なうことができないという事態になる。本実施の形態は、こうした問題★を回避して、ネットワーク中継プロセスの数のかかわらず、着実にプリフェッチが行なえるように次のような方策をとっている。すなわち、プリフェッチ子プロセスを起動するか否かを判断するにあたって、「その時点でのすべてのproxy ネットワーク中継個数」に「一定数MAXPROCESS」を加えたものを新たなMAXPROCESSとして使用したうえで上の数11の判断を行なっている。これにより、次のstep11以後のループを開始した時点ですべてのネットワーク中継プロセスの合計数が初期定数MAXPROCESSを超えていても、上のようにして計算された新しい定数MAXPROCESSが実際の比較の際には使用されるので、プリフェッチ用の並列子プロセスの新たな起動が可能となる。

【0381】以下具体的な実施方法について説明する。まず、ある時点でのすべてのproxyによるネットワーク中継個数をNUMPROCESSとする。NUMPROCESSは、OSのプロセステーブルから得ることができる。UNIX OS の下であれば次に挙げるコマンドの組合せでNUMPROCESSを計算することができる。

【0382】

【表54】

/usr/bin/ps	プロセステーブルの標準出力へ出力
/usr/bin/awk	特定フィールドの切出し
/usr/bin/grep	中継子プロセス名称に合致するものを取出す
/usr/ucb/wc	行数を数える

【0383】これらコマンドをたとえば次のように組合せることでNUMPROCESSを得ることができる。 ☆

☆【0384】

【数13】

71　　72

```
NUMPROCESS = '/usr/bin/ps -ax | awk '{print $5}' |
             grep ^DeleGate | /usr/ucb/wc -l '
```

【0385】ここでは、ネットワーク中継プロセス名称が既に述べたように"DeleGate"であることを利用している。さらに次の式によりMAXPROCESSを再計算する。　*

```
MAXPROCESS = MAXPROCESS + NUMPROCESS
```

*【0386】
【数14】

【0387】これにより新たなMAXPROCESSを得ることができる。
step12
図9に示されるstep12.1とstep12.2とをここではまとめてstep12と呼ぶことにする。step12.1以降の処理は、step10でインターリーブ化したプリフェッチリスト"prefetch _url.txt"の各行を1行ずつメモリ上の行バッファURLBUFに読込み、読込まれたURL に従ってプリフェッチ子プロセスを起動するループ処理である。このとき、第2のproxy を用い、当該ファイルオブジェクトをプリフェッチするためのプリフェッチ子プロセスをバックグラウンドプロセスとして1つ起動する。
【0388】なお、ネットワークアクセスをするためには、プリフェッチ子プロセスから直接httpプロトコルを発生させればよい。そのために専用のプログラムを作成しておけばよい。このプログラムの名称を仮にwebacces※

※s とする。
【0389】プログラムwebaccess は、UNIX OS のネットワーク機能の基本であるソケット機構を利用するものである。proxy プロセスのアドレスと、TCP/IPで使用されるポート番号とを指定して、プロセス間数通信をするためのネットワーク接続を行なう。さらにMAXPROCESSは、httpプロトコルにおけるGET コマンドを直接proxyサーバに書込み、さらに応答データをメモリに読み捨てる。読み捨てられた応答データは、proxy サーバのキャッシュ機構により自動的にキャッシュファイル16（図9参照）に蓄積される。
【0390】webaccess を使用する場合の実際の記述例は次のとおりである。
【0391】
【数15】

```
webaccess proxy:port http://xxx.co.jp/text/index.Html
```

【0392】この場合webaccess に対して、proxy サーバのアドレスと、使用されるポート番号とが指定される。webaccess はこれに応答して上記記述中に含まれるURL にアクセスしファイルを読出して内部のメモリに読み捨てる。proxy プロセスが利用しているマシンのネッ★

★トワークアドレスがproxyserver 、TCP/IPのポート番号が10001 番とする。行バッファURLBUFに次のファイルオブジェクト名称が格納されているものとする。
【0393】
【数16】

```
URLBUF = http://www.xxx.co.jp/text/index.html
```

【0394】この場合次の命令によりwebaccess を起動させれば、当該proxy プロセス経由で、サーバ計算機のファイルオブジェクト"http://www.xxx.co.jp/test/index.html"にアクセスできる。このとき中継子プロセスで☆

☆あるproxy 子プロセスができる。
【0395】
【数17】

```
webaccess proxyserver:10001 $URLBUF &
```

【0396】本実施の形態では、数15により記述される、ファイルオブジェクト（URL ）についてのキャッシュ更新アクセスにより、第1のproxy プロセスの管理するキャッシュファイル16を最新状態に保つことができる。なお数17で「&」が末尾に付けられているのは、子プロセスをバックグラウンドで並列に動作させるためである。
【0397】step13
続いて図9のstep13で、プリフェッチ子プロセスの数と第1のproxy プロセス14のネットワーク中継子プロセスの数とを計測し加算して、変数processes に加える。これらプロセス数も、step11で行なわれたのと同様、プロセステーブルから得ることができる。
【0398】step14
続いてstep14で、上述のようにして得られた変数proces◆

◆ses が、step11で計算された最大プロセス数MAXPROCESS以下か否かについて判定する。以下であれば制御はstep2.1 に戻り、プリフェッチアクセスリストの次の1行を抽出し、さらにバックグラウンドでプリフェッチ子プロセスを起動する。
【0399】step15
step14で、変数processes が最大プロセス数MAXPROCESSよりも大きいと判定されると、処理を一定時間休止（たとえば10秒）する。この一定時間の休止を制御するとstep13に戻し、プリフェッチ子プロセスの並列起動の起動を再び試みる。
【0400】以上のようなループ処理によって、プリフェッチ子プロセスの個数と、第1のproxy プロセス14を利用しているクライアント計算機ユーザのネットワーク中継子プロセスの個数との合計を制限しながら、プリ

フェッチ子プロセスの起動を制御する。この場合第1のproxy プロセス14を利用しているユーザによるネットワークアクセスの子プロセスの数がMAXPROCESS以上であったときにプリフェッチ子プロセスが起動待ちとなる。

【0401】したがって、クライアント計算機のユーザが第1のproxy プロセス14を利用している時間帯にプリフェッチプロセスを実行しても、プリフェッチ子プロセスには低い優先権しか与えられない。そのためプリフェッチプロセスの始動を内部ユーザの活動時間帯に始めても内部ユーザのネットワーク利用の妨げとはならない。また逆にプリフェッチプロセスが、クライアント計算機のユーザの活動時間の先頭部分に多少食い込んでも内部ユーザのネットワーク利用にはそれほど支障がない。

【0402】したがって、プリフェッチプロセスの開始から終了までより長時間のプリフェッチ処理を行なうことが可能になる。より多くのファイルオブジェクトのプリフェッチができるので、キャッシュデータの量を増やすことができる。結果としてキャッシュヒット率を高めることができる。これは、実施の形態1において既に述べたのと同様に、平均ファイルオブジェクト転送速度を増加させるという効果をもたらす。見方を変えれば、同じ平均ファイルオブジェクト転送速度を維持しながら、内部ネットワークのより多くのユーザがネットワーク利用をできるようにするという効果がある。

【0403】なお、図9のstep15の後に、このプリフェッチプロセスループを指定時刻に強制終了するためにOSタイマをセットする。OSタイマは、UNIX OS に標準的に提供されているコマンドを利用して、指定時間に指定の動作を実行させることで実現できる。指定時刻に上記したプロセスを終了するためには、具体的には次のようにすればよい。図8および図9に示されるプリフェッチプロセスには、実行時にはプロセスID番号がシステムから与えられる。そこでこのプロセスID番号を使用して、UNIX OS のkill命令を次のようにOSに与える。

【0404】

【数18】

```
echo "kill -9 プロセス ID 番号" | at 8:00    *
```

*【0405】このコマンドは、プロセスID番号で指定されたプロセスを終了させるためのkill命令を、8時00分に発行するようにOSに対して予約するものである。

【0406】以上は、プリフェッチプロセスの詳細な説明である。次に、第1のproxy プロセス14および第2のproxy プロセス34におけるキャッシュ管理手法について説明する。本発明では、2つのproxy プロセスが図1、図9等に示すようにキャッシュファイル16を共有している。しかも両者ともキャッシュファイル16を、ファイルオブジェクト単位で更新する。そのためキャッシュファイル16について、ファイルオブジェクト単位で排他制御ができなければ、ネットワーク中継子プロセス同士で競合が生じてしまう。そのためproxy プロセスのキャッシュ管理手法が排他制御を備えている必要がある。

【0407】本願の実施の形態では、公知の"DeleGate"と呼ばれるproxy ソフトウェアを使用している。このソフトウェアは排他制御機能を備えており、上述したような本願発明の実施の形態を実現するに十分な機能を備えている。

【0408】DeleGateにおけるキャッシュ管理手法は、ソースコードの形で公開されている。したがって以下このソースコードに沿ってその手法について述べる。なお本願発明を実現するためのproxy サーバソフトウェアとしては、このDeleGateに限定されるものではなく、同様のキャッシュ管理手法が使われるキャッシュ機能付proxy サーバソフトウェアであって排他制御機能を有するものであれば利用することができる。

【0409】既に述べた例に従って、proxy プロセスがproxyserver という名称のマシン（システム）で動作しており、TCP/IPのポート番号10001 番でURL 取得要求を受付けるものとする。このとき、バッファURLBUFに、指定のURL 表記が格納されているものとする。すると、たとえば図9に示すプリフェッチプロセス41は次のコマンドを実行する。

【0410】

【数19】

```
webaccess proxyserver:10001 $URLBUF
```

※数14の実行を待ち受ける。第2のproxy プロセス34により生成された子プロセスは、以下のような処理を実行する。なおproxy プロセスは、キャッシュファイルとして図7に示すキャッシュファイル16を持っている。このキャッシュファイル16は、通常のUNIXファイルシステムの一部として作られる。この場合、キャッシュ用ディレクトリとして/cacheというパーティションを使用するものとする。このとき、バッファURLBUFに格納されている文字列のうち次の文字列

【0411】なお第2のproxy プロセス34とプリフェッチプロセス41との間の通信はTCP/IPを用いて行なわれる。例としてバッファURLBUFに"http://www.sharp.co.jp/sample/test.html" という文字列が格納されているものとする。

【0412】プリフェッチプロセス41が数14に示すコマンドを実行することにより、URL 取得要求が第2のproxy プロセス34に出される。第2のproxy プロセス34はこれに応答して子プロセスを生成し、実際のファイルオブジェクトの取得処理をその子プロセスに任せて、自プロセスは再びプリフェッチプロセス41による※

【0413】

【数20】

75　　76

http://www.sharp.co.jp/sample/test.html

【0414】は、プロトコル名称がhttp、サーバ計算機名称がwww.sharp.co.jp 、ファイルオブジェクトの実体部分のディレクトリおよびファイル名称はsample/text.htmlである。このとき、キャッシュファイル16は次の＊名称規則に従ってキャッシュファイルを生成しようとする。

【0415】

【数21】

/cache/プロトコル名称/サーバ計算機名称/ファイルオブジェクトの実体部分

【0416】したがって"http://www.sharp.co.jp/sample/test.html" のディレクトリおよびファイル名は次のようにして指定される。

※【0417】

【数22】

※10

/cache/http/www.sharp.co.jp/sample/test.html

【0418】したがって、図10を参照して、proxy プロセスは次のようにキャッシュファイル16の排他制御を行なう。

【0419】（1） まずキャッシュファイル16の更新ルーチンを開始すると（A1）、指定されたURL を上記した規則に従ってキャッシュファイル名称に変換する。今述べている例では上述の数22というファイル名称が得られる（A2）。そしてこのようにして変換されたファイル名称がキャッシュファイル16内に存在するか否かを調べる。具体的には、OSのopen（）システムコールによって、このファイルを既存のファイルとしてオープンすることができるかどうかを調べることによりこのファイルの存在を調べる（A3）。

【0420】（2） このファイルが存在しなければ（A3でNO）、このキャッシュファイルをファイル新規作成モードでオープンする（A4）。

【0421】（3） さらにこのキャッシュファイルに対して他のプロセスから書込が行なえないように排他ロックする（A5）。これ以後の処理が排他制御である。

【0422】（4） 次にネットワーク経由で、バッファURLBUF内の文字列で指定されたファイルオブジェクトを、指定されたサーバ計算機から取得する（A6）。

【0423】（5） このようにして取得したファイルオブジェクトをキャッシュ更新プロセスに渡すとともに、キャッシュファイル16に書込む（A7）。このときのキャッシュファイル名称は数22に示したとおりである。

【0424】（6） 書込が完了するとキャッシュファイルをクローズして排他ロックを解除する（A8）。

【0425】ステップA3でキャッシュファイルがオープンできた場合、すなわち指定されたキャッシュファイルが既に存在した場合には次のような処理を行なう（A9）。

【0426】（7） キャッシュファイル/cache/http/www.sharp.co.jp/sample/test.htmlの最終変更時刻TmをOSから得る。UNIXであれば、最終変更時刻はfstat（）システムコールで得られる（A10）。fstat（）システムコールは、過去から未来に向かって時間の経過とともに単純に増加する関数である。

★【0427】（8） 続いて現在時刻をシステムコールを用いて読出しTnowという変数に格納する（A11）。

【0428】（9） 上記したキャッシュファイルの最終変更時刻Tmと現在時刻Tnowとを比較し、その差がキャッシュ有効期限以内かどうかを判定する（A12）。

【0429】（10） キャッシュ有効期限以内であれば（A12においてYES）、このキャッシュファイルを共有排他ロックする。すなわちこのキャッシュファイルに対して、他のプロセスからの書込はできないが、読出はできるようにする（A13）。

【0430】（11） 続いて現在のキャッシュファイルを読出し、キャッシュ更新プロセスに転送する（A14）。

【0431】（12） キャッシュファイルをクローズし、排他制御のロックを解除する（A8）。

【0432】一方、キャッシュ有効期限を過ぎている場合（A12においてNO）、次のような処理を行なう。

【0433】（13） キャッシュファイルを排他ロックし、他のプロセスからの書込が行なわれないようにする（A15）。

【0434】（14） 次にネットワーク経由で、バッファURLBUFに格納された文字列で指定されたファイルオブジェクトをサーバ計算機から取得する（A16）。

【0435】（15） こうして取得したファイルオブジェクトをキャッシュ更新プロセスに渡すとともに、数22で示されるキャッシュファイルに書込む（A17）。

【0436】（16） 続いてキャッシュファイルをクローズし排他ロックを解除する（A18）。

【0437】以上が、上述の実施の形態で使用されているproxy プロセスのキャッシュ制御手順である。本願発明の各実施の形態では、このようなproxy プロセスによる排他制御機能を利用して、キャッシュファイルシステム（たとえば図7のキャッシュファイル16）を、第1のproxy プロセス14と第2のproxy プロセス34とで共有している。なお第2のproxy プロセス34は、既に述べたようにキャッシュ有効期限＝0である。またキャッシュファイルの最終変更時刻Tmは常に現在時刻Tnowよりも小さい。したがって、図10のステップA12で行

なわれる判断は、第2のproxy プロセス34経由のサーバ計算機アクセスでは必ずNOという結果になり、キャッシュファイルの更新（A16）が必ず行なわれる。

【0438】以上のようにこの実施の形態でも、ファイルオブジェクト（URL ）のプリフェッチにより、第1のproxy プロセス14の管理するキャッシュファイル16の容量をn×24時間／m倍に増やし、かつ最新状態に保つことが可能である。

【0439】なお本実施の形態のstep5でn日分の過去のアクセスログファイルを集めたが、このn日という数字は、キャッシュファイルの最大容量とファイル転送実績とから決められるものであって、システムごとに異なる。1GBの容量を持つキャッシュファイル装置に、1日当り100MBのファイルオブジェクトが蓄積されるようであれば、n＝5程度、すなわちファイルオブジェクトが500MB程度プリフェッチされるような設定とすれば、キャッシュファイル16が溢れることはない。

【0440】このようにキャッシュファイルの蓄積の実績から、キャッシュファイルを溢れさせないnという数字を算出するのは容易である。そのためには次のようにする。たとえば1GBの容量を有するキャッシュファイル装置であれば、キャッシュ蓄積量の最小値MIN＝60%、最大値MAX＝80%と定めておく。キャッシュ蓄積の実績量がMINを下回るようであれば、毎日のプリフェッチ起動時にn＝n＋1とする。またキャッシュ蓄積の実績量がMAXを上回る場合には、n＝n－2などと減らす。こうすることでキャッシュ蓄積容量をほぼMIN、MAXの間に保つことができる。

【0441】このようなプリフェッチプロセスは、内部ネットワークから外部ネットワークへのクライアント計算機ユーザのアクセスがあまりない時間帯を利用して起動することができる。そうすることによってプリフェッチプロセスによるファイルオブジェクトアクセスが、内部ネットワークユーザの外部ネットワークアクセスの妨げにならないような運用をすることができる。上述の実施の形態では、プリフェッチプロセスはクライアント計算機の利用が少なくなる21時00分に起動し、翌日の8時00分には遅くとも終了させるようにしてある。8時00分にプリフェッチプロセスがまだ動作しているのであれば既に述べたように強制終了をさせている。これにより、プリフェッチリストが大きくて、プリフェッチに時間を要し、そのため翌日の8時までにプリフェッチプロセスが終了しない場合であっても、クライアント計算機の利用が始められる時間帯にはプリフェッチプロセスが確実に終了される。これ以後の、内部ネットワークユーザによるクライアント計算機の利用は妨げられない。

【0442】ところで、上述の実施の形態でも、第1のproxy プロセス14のキャッシュ有効期限M＝24時間としている。そのため夜間に更新されたキャッシュデータの内容は、更新後翌晩の21時までは少なくとも有効である。すなわち、キャッシュファイルのうち少なくとも変更頻度の高いファイルオブジェクトは過去24時間以内に更新された情報に維持される。一方で、キャッシュファイルの容量は1日の平均アクセス量のn日分である。キャッシュヒット率は、キャッシュの蓄積期間が24時間の場合と比較してn＝7とすると3倍以上となることが経験的にわかっているから、上述のような設定をすることでキャッシュファイルの内容を新鮮に保ちながらキャッシュヒット率を高めることができる。また、第1のproxy プロセス14のキャッシュ有効期限M時間以下の時間間隔でプリフェッチプロセスの起動が行なわれるから、キャッシュファイルは常に有効期限内に保たれる。したがってプリフェッチから次のプリフェッチまでの間、プリフェッチの効果が有効に保たれる。

【0443】［実施の形態7］実施の形態1～6では、ゲートウェイ計算機とクライアント計算機とが別々の計算機であるものとして説明した。しかしながら、図11に示されるように、ゲートウェイ計算機55の中にクライアントプロセス42を起動して、クライアント計算機の機能を含ませても何ら問題はない。その逆にクライアント計算機の中にゲートウェイ計算機の機能を含ませても支障がないことはもちろんである。なお図11において図7に示される各構成要素と同じ構成要素には同一の参照番号および名称を付してある。それらの機能も同一である。したがってここではそれらについての詳しい説明は繰返さない。

【0444】このようにクライアント計算機とゲートウェイ計算機との機能を同一の計算機内に組込むことにより、より多くのファイルオブジェクトのプリフェッチをすることが可能になる。そのためキャッシュデータの量を増やすことができるので、結果としてキャッシュヒット率を高めることができる。ゲートウェイ計算機として得られる効果は、既に述べた他の実施の形態の装置により得られるものと同様である。

【0445】再びここで各実施の形態の効果について述べる。実施の形態1によれば、内部ネットワークユーザが利用するproxy プロセスでは、キャッシュ有効期限Mがたとえば24時間というように短く設定されている。したがってキャッシュされたファイルオブジェクトの内容は、最近の24時間以内の内容であることが保証されている。すなわちキャッシュファイルのステールデータ率は小さく抑えられている。一方、キャッシュファイル自体にはn日分のキャッシュファイルが蓄積されている。キャッシュの蓄積量が、プリフェッチプロセスを行なわない場合と比較してn×24／M倍となるので、キャッシュのヒット率はキャッシュ有効期限をn日としたときに相当する高い値となる。

【0446】従来のproxy サーバ装置で、キャッシュ有効期限を24時間と設定した場合のキャッシュヒット率

は１２％程度であることが観察されている。一方実施の形態１のproxy サーバ装置によるプリフェッチ処理を実行した場合、ｎ＝７日とすると、上述のようにキャッシュヒット率が高くなることが予測され、実際にもヒット率は３５％となった。このためweb サーバへの問合せが減少し、応答速度が向上した。

【０４４７】実施の形態２のproxy サーバ装置では、プリフェッチを同時並列的に進めることで、プリフェッチの開始から終了までの時間を短縮することができる。指定終了時刻までの間に、より多くのファイルオブジェクトをプリフェッチできる。キャッシュデータの量を増やすことができるので、結果としてキャッシュヒット率を高めることができる。

【０４４８】実施の形態３のproxy サーバ装置では、プリフェッチアクセスリスト内のファイルオブジェクト名称をインターリーブしてからプリフェッチを同時並行的に進める。そのため特定のサーバ計算機に対してアクセスが集中することが避けられる。サーバ計算機の応答が遅くなってボトルネックとなることが避けられるので、プリフェッチの開始から終了までの時間を短縮できる。指定終了時刻までの間により多くのファイルオブジェクトのプリフェッチが可能になる。キャッシュデータの量を増やすことができるので、結果としてキャッシュヒット率を高めることができる。

【０４４９】実施の形態４によれば、内部のユーザが第１のproxy プロセスを利用している時間帯にプリフェッチプロセスを実行しても、プリフェッチ子プロセスには低い優先権しか与えられていないので、内部ユーザのネットワーク利用活動を妨げることがないという効果がある。プリフェッチプロセスを、内部ユーザの活動時間帯内に始めても、内部ユーザのネットワーク利用の妨げとはならない。またプリフェッチプロセスの終了を、内部ユーザの活動時間帯内までずれ込ませても内部ユーザのネットワーク利用の妨げとはならない。開始から終了までより長時間のプリフェッチ時間を利用することが可能になり、より多くのファイルオブジェクトをプリフェッチすることができる。キャッシュデータの量も応じて増加できるので、結果としてキャッシュヒット率を高めることができる。

【０４５０】実施の形態５のproxy サーバ装置では、キャッシュファイル中のファイルオブジェクトを消去するための期限はｎ日程度としている。この期限は、キャッシュファイルの有効期限よりも長く選択されており、有効期限が過ぎたキャッシュファイルも直ちには消去されない。そして有効期限が過ぎたファイルオブジェクトであっても、アクセス要求が生じ該当するサーバ計算機の元データの変更時刻を照会することにより、サーバ計算機の元データが変更されていなければ再びこのキャッシュされているファイルオブジェクトが有効となる。そしてこのファイルオブジェクトのヘッダ情報の変更時刻を更新するだけで、実際のファイルオブジェクトのデータ転送をすることなく実質的にファイルオブジェクトのプリフェッチを実現できる。そのため再ロードのためのトラフィックを削減することができる。

【０４５１】プリフェッチによるキャッシュのヒット率向上により、昼間のユーザの活動によるファイルオブジェクト転送量は十分減少しており、上述のプリフェッチを行なってもトラフィックがいたずらに増大するおそれはない。またキャッシュファイルの内容は、いずれも有効期限内に維持できるので、その内容の新鮮さを維持することができる。

【０４５２】実施の形態６では、第２のproxy のログファイル中で変更があったことを示す行のみが抽出され、第１のproxy のログファイルから抽出されたプリフェッチリストの先頭にマージされる。一般に、前回のプリフェッチ時に変更があったファイルオブジェクトは、変更頻度が高いということができる。したがってこれらを優先してプリフェッチすることで、限られた時間を有効に利用して、キャッシュファイルの状態を最新の状態に近く維持するとが可能になる。

【０４５３】上述した実施の形態のいずれも、既存のproxy プロセスにさらにプリフェッチプロセスを追加するだけで実現でき、既存のproxy プロセス自体には何らの変更も必要ではない。そのため容易にキャッシュファイルのヒット率が向上したproxy サーバ装置を提供できるという利点がある。なお、ＨＴＴＰプロトコルを含めた参考文献リストを以下に示す。

【０４５４】

【表５５】

- [Ref.1] 横塚実、『インターネットとはなにか』
  http://www.rwcp.or.jp/people/yokotuka/Internet_report.html
- [Ref.2] page 170,"キャッシュ利用でトラフィックを低減するプロキシサーバー" 『企業ユーザのためのインターネットハンドブック』日経コミュニケーション、ISBN4-822-1365-X, 日経BP社
- [Ref.3] WWWに関する参考文献
  http://www.w3.org/hypertext/WWW/Protocols/Overview.html
- [Ref.4] HTTPの仕様に関するインターネット上の文献
  http://www.w3.org/hypertext/WWW/Protocols/HTTP2.html
- [Ref.4の日本語資料] 日本語のHTTPプロトコル仕様書としては、木内貴弘(kiuchi@epistat.m.u-tokyo.ac.jp)氏翻訳の
  http://www.epistat.m.u-tokyo.ac.jp/internet/draft-fielding-http-spec-01-jp.txt、
  オリジナルファイル名：著者：T. Berners-Lee、R. T. Fielding、H. Frystyk Nielsen がある。
- [Ref.5] 電子技術総合研究所の佐藤氏の作成されたdelegateについては
  http://www.etl.go.jp:8080/etl/People/ysato@etl.go.jp/DeleGate/から情報が入手できる。
- [Ref.6] キャラクタベースのWWWクライアントlynx の解説は、
  http://www.icsd6.tj.chiba-u.ac.jp/lynx/
- [Ref.7] WWWシステム全般の情報は、 World Wide Web コンソーシアム
  http://www.w3.org/hypertext/WWW/TheProject/
- [Ref.8] CERN httpdに関しては、
  http://www.w3.org/hypertext/WWW/Daemon/
- [Ref.9] 大家隆弘(徳島大学),WWW多段proxy計画（案）
  http://www.orions.ad.jp/project/multiproxy/multiproxy-jp.html
- [Ref.10] CERN httpd でのキャッシュ制御解説
  http://info.cern.ch/hypertext/WWW/Daemon/User/Config/Caching.html
- [Ref.11] 東田学 ,対外接続のトラフィックを軽減するためのキャッシュ・サーバ導入について
  http://www.osaka-u.ac.jp/odins/document/proxy-jp.html
- [Ref.12] 佐藤豊、" 多目的プロキシ・サーバDeleGateの機能詳解" インターフェイス １９９５年９月 p130-p146

【図面の簡単な説明】

【図１】本発明に係るゲートウェイ装置の一例であるproxy サーバ装置の動作概念図である。

【図２】WWWサーバ内の情報頁をクライアントソフトウェアでブラウズしたときの表示画面を示す図である。

【図３】実施の形態１のゲートウェイ装置であるproxy サーバ装置で行なわれるプリフェッチプロセスの実現アルゴリズムを示すフローチャートである。

【図４】実施の形態２のゲートウェイ装置であるproxy サーバ装置で行なわれるプリフェッチプロセスの実現アルゴリズムを示すフローチャートである。

【図５】実施の形態３のゲートウェイ装置であるproxy サーバ装置で行なわれるプリフェッチプロセスの実現アルゴリズムを示すフローチャートである。

【図６】実施の形態４のゲートウェイ装置であるproxy サーバ装置で行なわれるプリフェッチプロセスの実現アルゴリズムを示すフローチャートである。

【図７】本発明の実施の形態６に係るproxy サーバ装置のブロック図である。

【図８】実施の形態６のproxy サーバ装置におけるプリフェッチプロセスの前半部のフローチャートである。

【図９】実施の形態６のproxy サーバ装置におけるプリフェッチプロセスの後半部のフローチャートである。

【図１０】実施の形態６のproxy サーバ装置でのキャッシュファイル更新子プロセスのフローチャートである。

【図１１】本願発明の実施の形態７のproxy サーバ装置＊のブロック図である。

【図１２】従来のゲートウェイ計算機の構成を示す図である。

【図１３】proxy サーバ装置の概念図である。

【図１４】proxy サーバ装置の回路構成の第１の例を示す図である。

【図１５】proxy サーバ装置の回路構成の第２の例を示す図である。

【図１６】proxy サーバ装置の回路構成の第３の例であって、本願発明の実施の形態proxy サーバ装置としても使用されるproxy サーバ装置の回路構成を示す図である。

【図１７】キャッシュヒット率と平均アクセス速度との関係を示すグラフである。

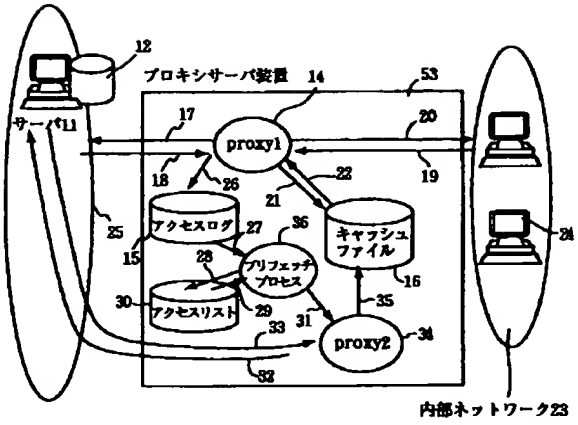
【図１８】キャッシュファイルサイズとキャッシュヒット率との関係を示すグラフである。

【図１９】キャッシュ有効期限とキャッシュファイルの大きさとの関係を示すグラフである。

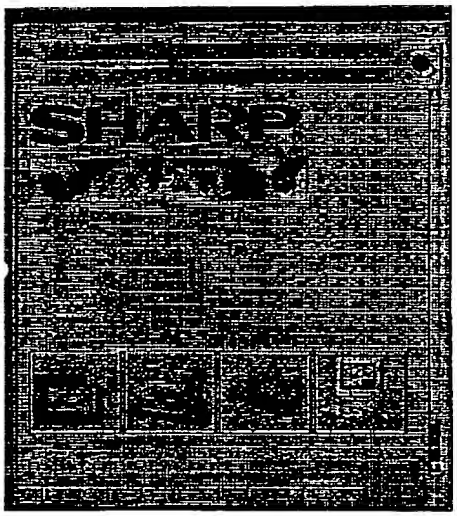
【符号の説明】

１１　サーバ計算機
１４　第1proxy プロセス
１５　アクセスログ
１６　キャッシュファイル
３０　アクセスリスト
３４　第2のproxy プロセス
３６　プリフェッチプロセス

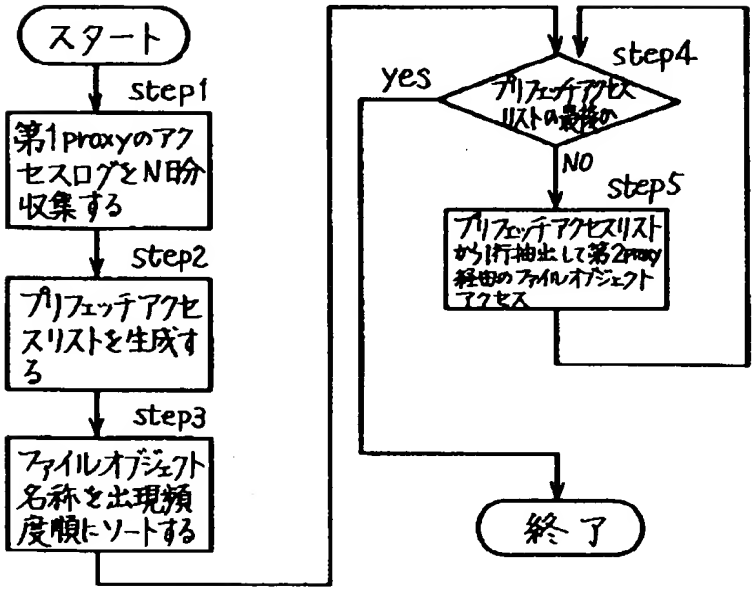
【図1】


12
プロキシサーバ装置 14
53
サーバ11
17
proxy1
20
18
26
22
19
21
アクセスログ
27
36
キャッシュファイル
25
15
28
プリフェッチプロセス
16
30
アクセスリスト
29
31
35
proxy2
34
33
32
24
内部ネットワーク23

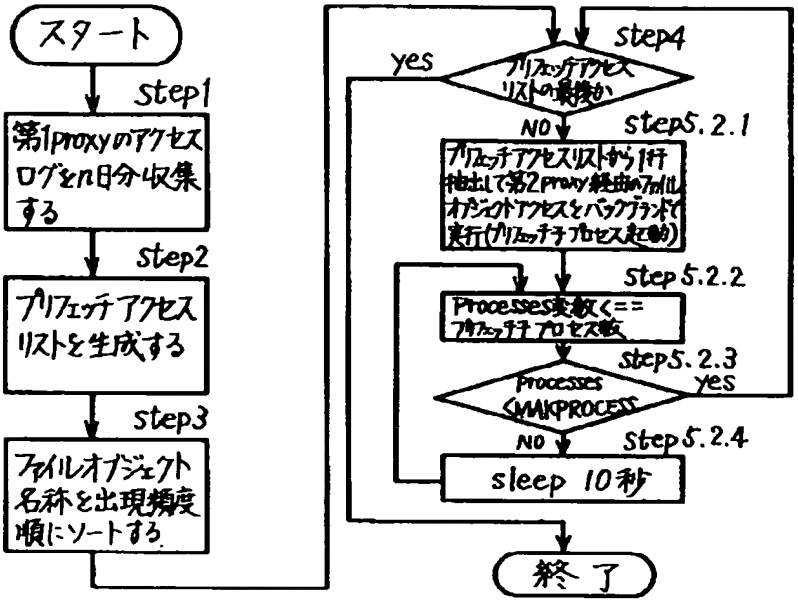
【図2】


SHARP

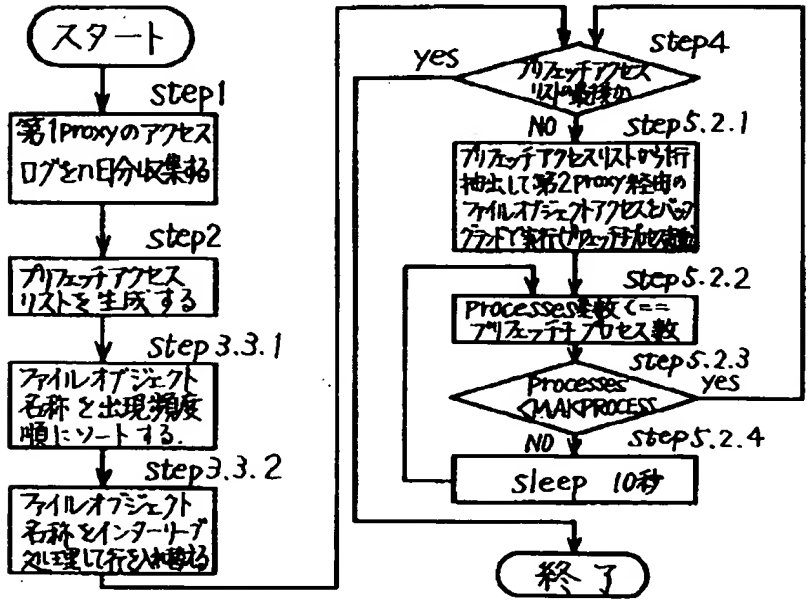
【図3】


スタート
step1
第1proxyのアクセスログをN日分収集する
step2
プリフェッチアクセスリストを生成する
step3
ファイルオブジェクト名称を出現頻度順にソートする
step4
yes
プリフェッチアクセスリストの最後か
NO
step5
プリフェッチアクセスリストから1行抽出して第2proxy経由のファイルオブジェクトアクセス
終了

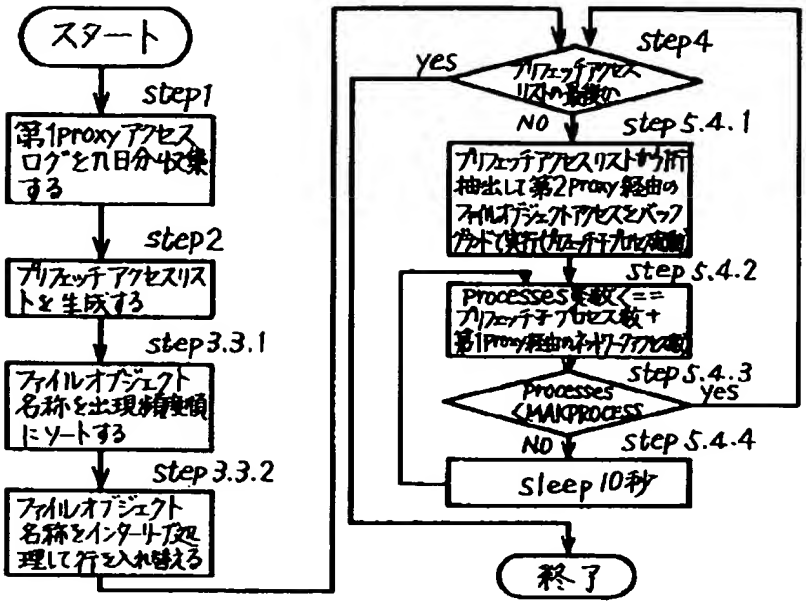
【図4】


スタート
step1
第1proxyのアクセスログをn日分収集する
step2
プリフェッチアクセスリストを生成する
step3
ファイルオブジェクト名称を出現頻度順にソートする
step4
yes
NO
step5.2.1
step5.2.2
Processes変数<==プリフェッチプロセス数
step5.2.3
Processes<MAXPROCESS
yes
NO
step5.2.4
sleep 10秒
終了

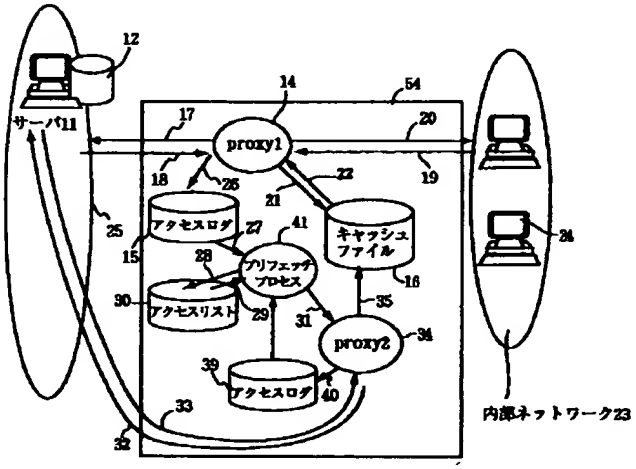
【図5】


スタート
step1
第1proxyのアクセスログをn日分収集する
step2
プリフェッチアクセスリストを生成する
step3.3.1
ファイルオブジェクト名称を出現頻度順にソートする.
step3.3.2
ファイルオブジェクト名称をインターリーブ処理して行う
step4
yes
NO
step5.2.1
step5.2.2
Processes変数<==プリフェッチプロセス数
step5.2.3
Processes<MAXPROCESS
yes
NO
step5.2.4
sleep 10秒
終了

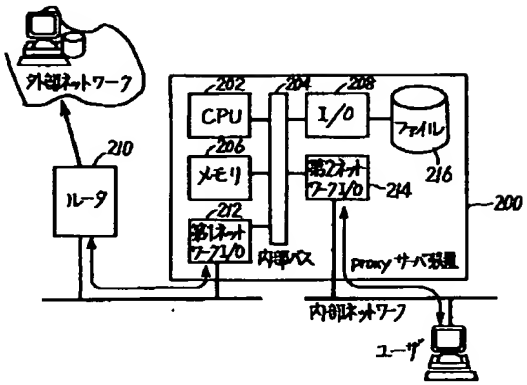
【図6】


スタート
step1
第1proxyアクセスログを1日分収集する
step2
プリフェッチアクセスリストを生成する
step3.3.1
ファイルオブジェクト名称を出現頻度順にソートする
step3.3.2
ファイルオブジェクト名称をインターリーブ処理して行を入れ替える
step4
yes
NO
step5.4.1
step5.4.2
step5.4.3
step5.4.4
Processes<MAXPROCESS
Sleep 10秒
終了

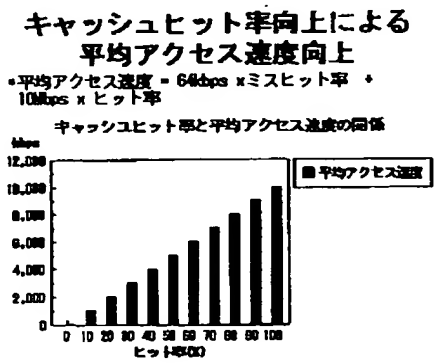
【図7】


サーバ11
proxy1
proxy2
アクセスログ
プリフェッチプロセス
キャッシュファイル
アクセスリスト
内部ネットワーク23

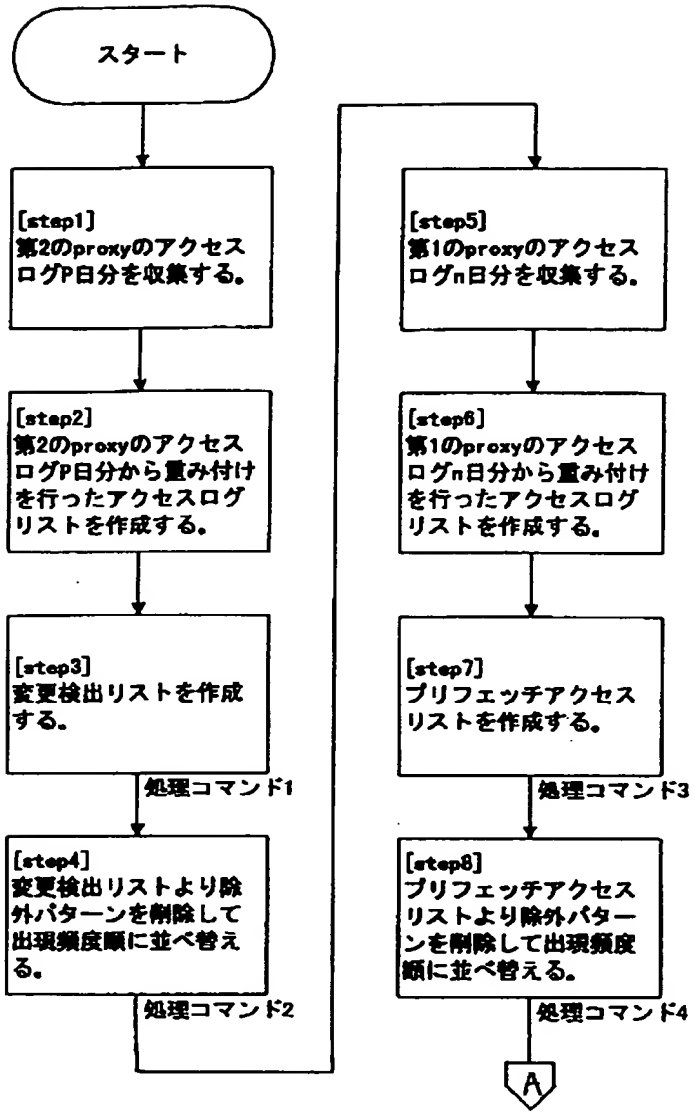
【図14】


外部ネットワーク
ルータ
CPU
I/O
ファイル
メモリ
第2ネットワークI/O
第1ネットワークI/O
内部バス
proxyサーバ装置
内部ネットワーク
ユーザ

【図17】

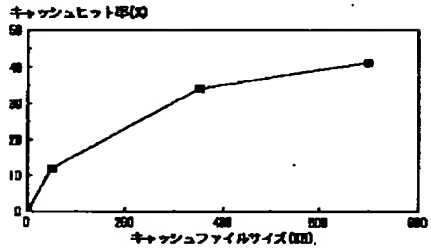

キャッシュヒット率向上による
平均アクセス速度向上
・平均アクセス速度＝64kbps×ミスヒット率＋10Mbps×ヒット率
キャッシュヒット率と平均アクセス速度の関係
kbps
12,000
10,000
8,000
6,000
4,000
2,000
0
0 10 20 30 40 50 60 70 80 90 100
ヒット率(%)
平均アクセス速度

【図8】



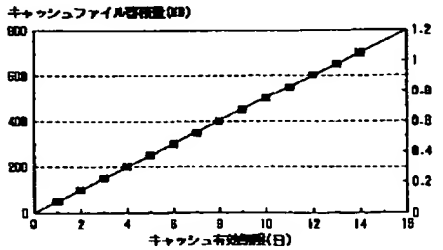
【図18】

キャッシュファイルサイズとキャッシュヒット率の関係

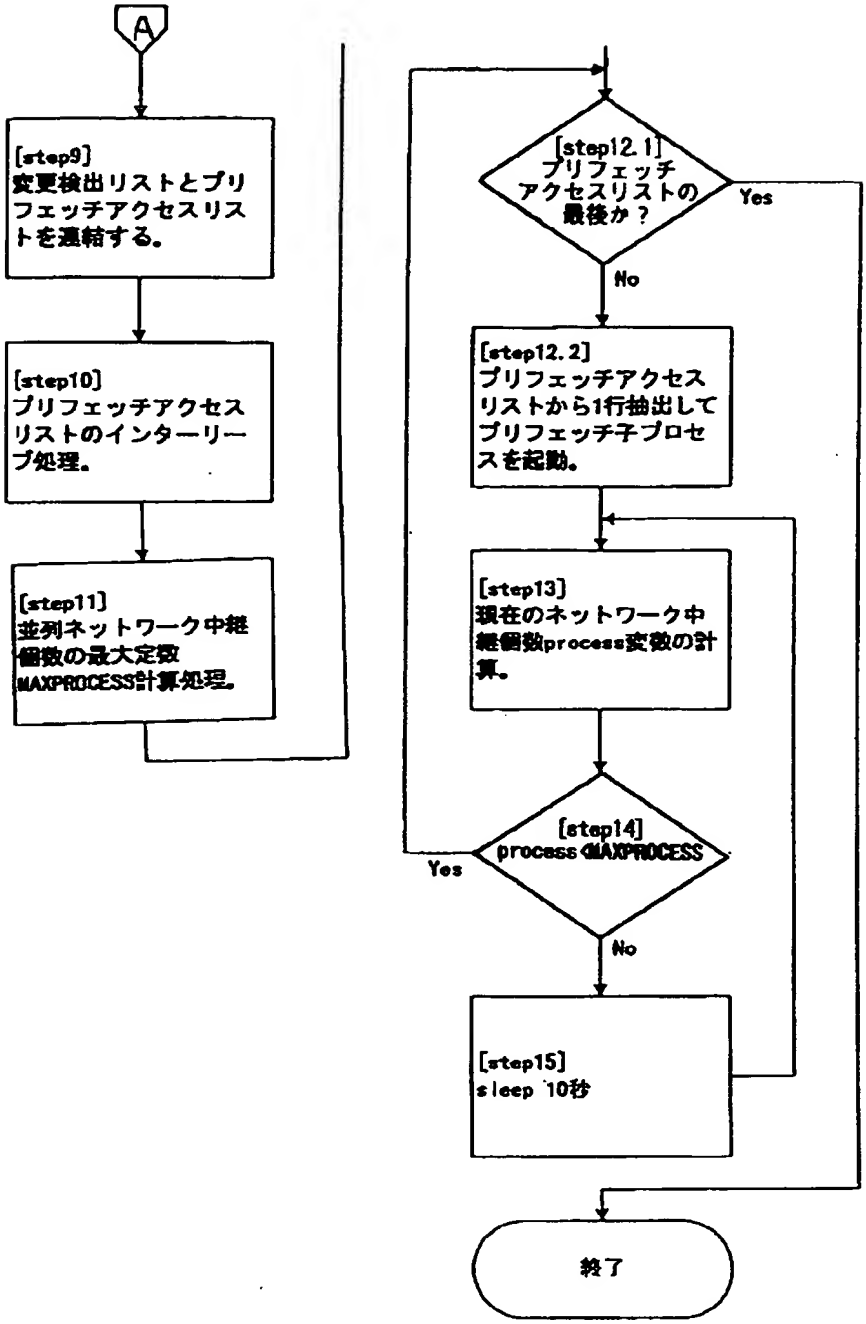


【図19】

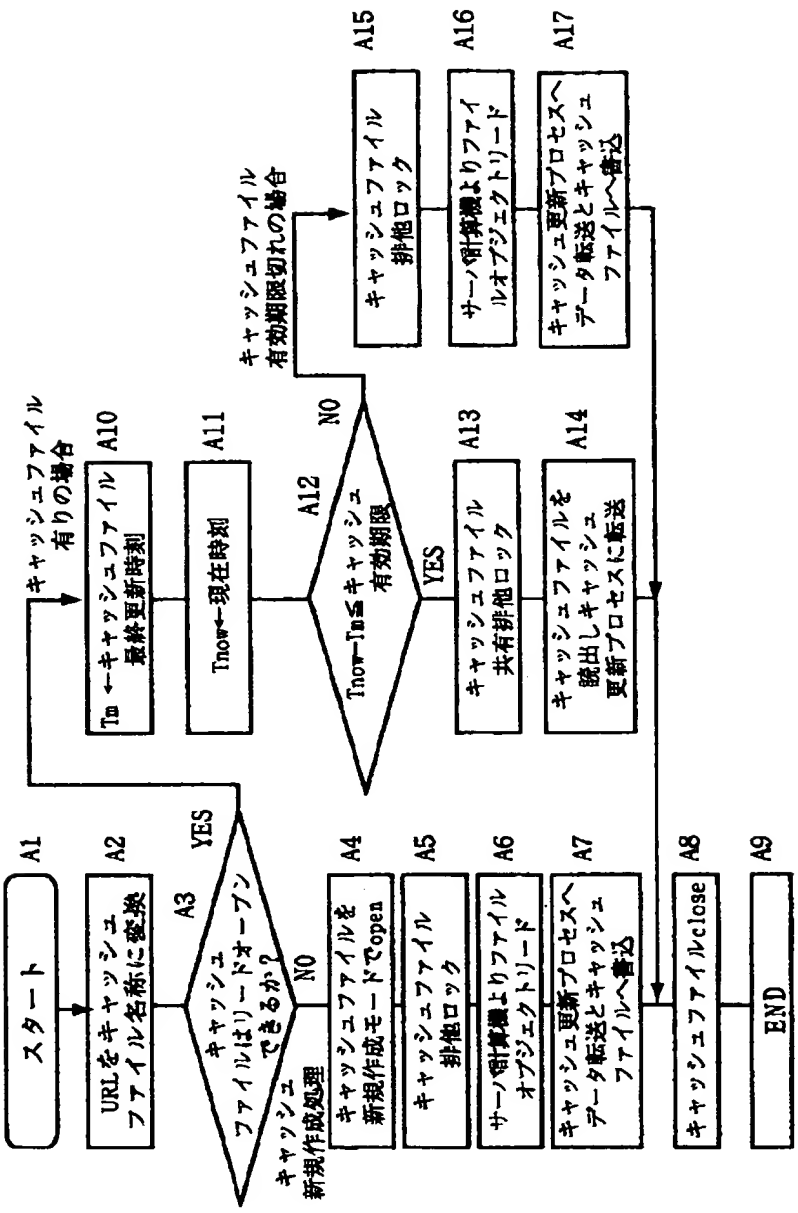
キャッシュ有効期限とキャッシュファイルの大きさの関係



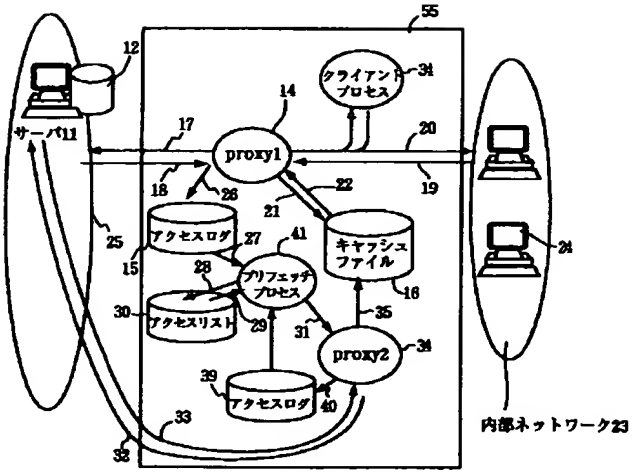
【図9】


A
[step9]
変更検出リストとプリフェッチアクセスリストを連結する。
[step10]
プリフェッチアクセスリストのインターリーブ処理。
[step11]
並列ネットワーク中継個数の最大定数MAXPROCESS計算処理。
[step12.1]
プリフェッチアクセスリストの最後か？
Yes
No
[step12.2]
プリフェッチアクセスリストから1行抽出してプリフェッチ子プロセスを起動。
[step13]
現在のネットワーク中継個数process変数の計算。
[step14]
process<MAXPROCESS
Yes
No
[step15]
sleep 10秒
終了

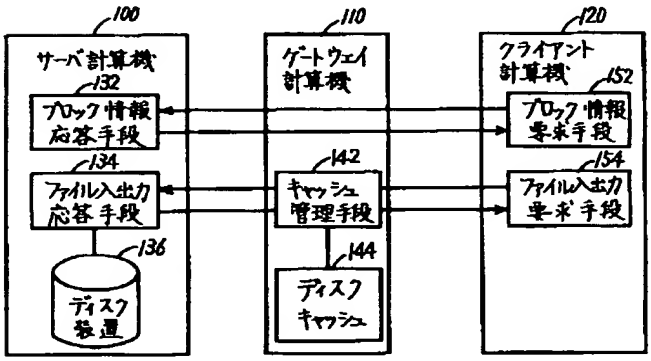
【図10】


スタート
A1
URLをキャッシュファイル名称に変換
A2
キャッシュファイルはリードオープンできるか？
A3
YES
NO
キャッシュ新規作成処理
キャッシュファイルを新規作成モードでopen
A4
キャッシュファイル排他ロック
A5
サーバ計算機よりファイルオブジェクトリード
A6
キャッシュ更新プロセスへデータ転送とキャッシュファイルへ書込
A7
キャッシュファイルclose
A8
END
A9
キャッシュファイル有りの場合
Tm←キャッシュファイル最終更新時刻
A10
Tnow←現在時刻
A11
Tnow−Tm≦キャッシュ有効期限
A12
NO
YES
キャッシュファイル共有排他ロック
A13
キャッシュファイルを読出しキャッシュ更新プロセスに転送
A14
キャッシュファイル有効期限切れの場合
キャッシュファイル排他ロック
A15
サーバ計算機よりファイルオブジェクトリード
A16
キャッシュ更新プロセスへデータ転送とキャッシュファイルへ書込
A17

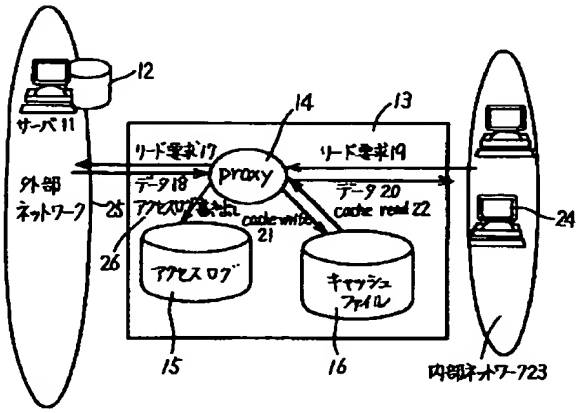
【図11】



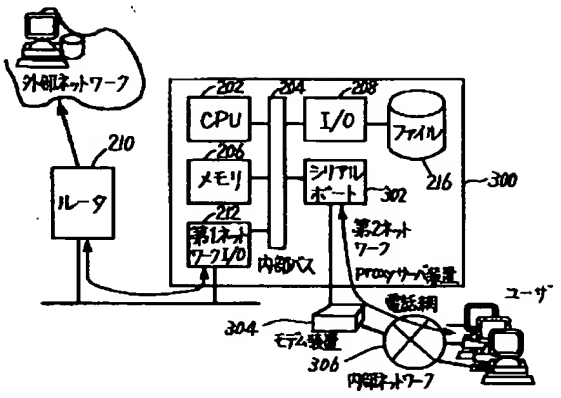
【図12】



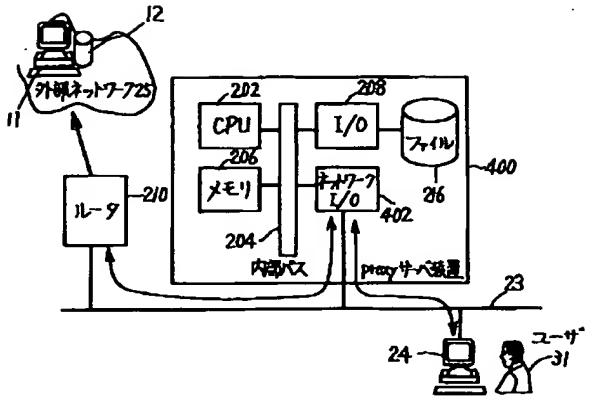
【図13】



【図15】



【図16】


12
外部ネットワーク
11
202
CPU
208
I/O
ファイル
400
206
メモリ
ネットワーク I/O
402
216
210
ルータ
204
内部バス
proxyサーバ装置
23
24
ユーザ
31